IBM MobileFirst Platform Foundation V8.0.0

Note

For up-to-date product documentation, see the IBM MobileFirst Foundation Developer Center.

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

IBM MobileFirst Platform Foundation V8.0.0

This edition applies to version V8.0.0 of IBM MobileFirst Platform Foundation and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition was updated last on 24 May 2017.

Contents

IBM MobileFirst Platform Foundation V8.0.0 documentation 1-1

Product overview 2-1
Product main capabilities 2-1
Product components 2-3
System requirements 2-7
Licensing in MobileFirst Server 2-8
Downloading IBM MobileFirst Platform Foundation V8.0.0 2-9
Matrix of features and platforms. 2-10
Accessibility features for IBM MobileFirst Platform Foundation 2-10

Release notes 3-1
What's new in V8.0.0 3-1
- What's new in building apps 3-1
- What's new in MobileFirst APIs 3-3
- What's new in MobileFirst security 3-6
- What's new in operating system support . . . 3-9
- What's new in deploying and managing apps 3-10
- What's new in MobileFirst Server 3-12
- What's new in IBM MobileFirst Analytics. . . 3-13
- What's new in push notifications 3-14

What's new in V8.0.0 interim fixes 3-15
Licensing 3-16
Web applications 3-16
Adapters 3-16
Cordova applications 3-17
Deprecated features and API elements. 3-17
Discontinued features and API elements 3-19
Known issues 3-25
Known limitations 3-25

Tutorials and additional resources . . 4-1

Upgrading to IBM MobileFirst Platform Foundation V8.0.0 5-1
Migrating apps from earlier releases 5-1
- Client API changes in V8.0.0 5-4
- Server-side API changes in V8.0.0 5-14
- Migrating client applications to IBM MobileFirst Platform Foundation V8.0.0 . . . 5-17
- Migrating existing adapters to work under MobileFirst Server V8.0.0 5-51
- Migrating to push notifications from event source-based notifications 5-54
- Migrating apps storing mobile data in Cloudant with IMFData or Cloudant SDK . . 5-80

Applying a fix pack to IBM MobileFirst Platform Server 5-104

Installing and configuring server-side components 6-1
Installation overview 6-1
Installing IBM MobileFirst Platform Server. . . . 6-2
- MobileFirst Server overview 6-2
- Tutorials about MobileFirst Server installation 6-4
- Installing MobileFirst Server for a production environment 6-39
- Installing and configuring for token licensing 6-150

Configuring MobileFirst Server 6-164
- Endpoints of the MobileFirst Server production server 6-164
- Configuring MobileFirst Server to enable TLS V1.2 6-165
- Configuring user authentication for MobileFirst Server administration 6-166
- List of JNDI properties of the MobileFirst Server web applications 6-171
- Configuring data sources 6-190
- Configuring logging and monitoring mechanisms 6-194
- Configuring license tracking. 6-195
- WebSphere Application Server SSL configuration and HTTP adapters 6-197

Installing and configuring the MobileFirst Analytics Server 6-198
Installing and configuring the Application Center 6-198
- Installing Application Center with IBM Installation Manager 6-199
- Installing the Application Center with Ant tasks. 6-204
- Manually installing Application Center . . . 6-207
- Configuring Application Center after installation 6-233

Installation reference 6-268
- Ant `configuredatabase` task reference . . . 6-268
- Ant tasks for installation of MobileFirst Operations Console, MobileFirst Server artifacts, MobileFirst Server administration, and live update services 6-274
- Ant tasks for installation of MobileFirst Server push service 6-287
- Ant tasks for installation of MobileFirst runtime environments 6-293
- Ant tasks for installation of Application Center 6-304
- Ant tasks for installation of MobileFirst Analytics 6-309
- Internal runtime databases 6-315
- Sample configuration files 6-318
- Sample configuration files for MobileFirst Analytics 6-319

Developing applications 7-1
Development concepts and overview 7-2

Applications 7-2
MobileFirst Server 7-4
Adapters 7-6
MobileFirst Operations Console overview . . . 7-6
Client app development environments 7-8
Setting up the development environment 7-9
Getting started with MobileFirst development 7-9
The IBM MobileFirst Platform Foundation Developer Kit 7-9
Setting up the MobileFirst Development Server 7-12
The MobileFirst command-line interface (CLI) 7-13
Setting up an internal Maven repository for offline development 7-23
Developing the client side of a MobileFirst application 7-24
Developing MobileFirst applications 7-24
Getting started with a sample MobileFirst application 7-25
Acquiring the MobileFirst SDK from the MobileFirst Operations Console 7-26
Developing native applications for iOS in Xcode. 7-27
Developing native applications in Android Studio 7-52
Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal 7-65
Developing web applications 7-73
Developing Cordova applications 7-83
JSONStore 7-134
Certificate pinning 7-185
Developing the server side of a MobileFirst application 7-187
Adapters overview 7-187
Adapters as Apache Maven projects 7-189
MobileFirst Java adapters. 7-192
MobileFirst JavaScript adapters 7-204
Configuring adapters 7-227
Tools for testing and debugging adapters 7-230
Client access to adapters 7-231
Troubleshooting an error when an application or an adapter is pushed to a MobileFirst Server 7-233
Updating Cordova client apps directly 7-235
The Direct Update lifecycle 7-236
Creating and deploying updated web resources to MobileFirst Server 7-237
Implementing secure Direct Update on the client side 7-239
Default Direct Update user interface 7-240
Serving Direct Update requests from a CDN 7-241
Customizing the Direct Update user interface and process 7-244
Push notification 7-248
Push notification architecture 7-250
Getting started with push notifications . . . 7-250
Security for push notification clients 7-251
Setting up push notifications 7-253
Broadcast notifications. 7-256
Tag-based notifications 7-257
Unicast notifications 7-257
Sending push notifications 7-258
Sending SMS notifications 7-264
REST Services APIs. 7-264
Troubleshooting push notification problems 7-265
MobileFirst security framework. 7-265
Overview of the MobileFirst security framework 7-265
OAuth resource protection 7-271
Confidential clients 7-279
Security checks 7-281
Access tokens. 7-303
Client security APIs 7-305
Configuring IBM WebSphere DataPower as the OAuth authorization server. 7-314
Configuring the MobileFirst Server keystore 7-316

API reference 8-1
MobileFirst client-side API 8-1
JavaScript client-side API 8-1
JavaScript client-side push API. 8-4
JavaScript web analytics client-side API. . . . 8-4
Objective-C client-side API for iOS apps . . . 8-5
Objective-C client-side push API for iOS apps 8-5
Objective-C client-side API for hybrid apps . . 8-5
Java client-side API for Android apps 8-5
Java client-side push API for Android apps . . 8-6
C# client-side API for Windows 10 Universal Windows Platform and Windows 8 Universal apps 8-6
C# client-side push API for Windows 10 Universal Windows Platform and Windows 8 Universal apps 8-6
MobileFirst server-side API 8-6
JavaScript server-side API 8-7
Java server-side API 8-7
MobileFirst Java Token Validator API 8-7
REST API for the MobileFirst Server administration service 8-7
Adapter (GET) 8-7
Adapter (DELETE) 8-10
Adapter (POST) 8-13
Adapters (GET) 8-17
Adapter Configuration (GET). 8-20
Adapter configuration (PUT) 8-22
Application Authenticity (DELETE). 8-27
Application Configuration (GET) 8-30
Application Configuration (PUT) 8-32
Application Descriptor (GET). 8-37
Application Environment (GET) 8-39
Application (GET) 8-41
Application (POST) 8-43
Applications (GET) 8-48
Application License Configuration (POST) . . 8-51
Application license configuration (GET) . . . 8-54
Application Version (GET). 8-56
Application Version (DELETE) 8-58
Audit (GET) 8-61
Confidential Clients (GET). 8-62
Confidential Clients (PUT). 8-64
Create Subscription (POST) 8-68
Create Tag (POST) 8-69

Delete APNs settings (DELETE) 8-70
Delete GCM settings (DELETE) 8-71
Delete WNS settings (DELETE) 8-72
Delete Message (DELETE) 8-73
Delete Subscription (DELETE) 8-74
Delete Tag (DELETE) 8-75
Deploy (POST) 8-76
Deploy Application Authenticity Data (POST) 8-80
Deploy a web resource (POST) 8-83
Device Application Status (PUT) 8-86
Device Status (PUT) 8-90
Device (DELETE) 8-93
Devices (GET) 8-96
Diagnostic Service (GET) 8-100
Export adapter resources (GET). 8-103
Export adapters (GET). 8-105
Export application environment (GET) . . . 8-106
Export application environment resources (GET) 8-107
Export application resources (GET) 8-109
Export applications (GET) 8-110
Export resources (GET) 8-111
Export runtime resources (GET) 8-113
Farm topology members (GET) 8-114
Farm topology members (DELETE) 8-116
Get Message (GET) 8-118
Get Tags (GET) 8-119
Get APNs Settings (GET) 8-122
Get GCM Settings (GET) 8-123
Get WNS Settings (GET) 8-125
Global Configuration (GET) 8-127
Keystore (GET) 8-129
Keystore (POST). 8-130
Keystore (DELETE). 8-133
License configuration (DELETE) 8-136
Push Device Registration (GET) 8-139
Push Device Registration (DELETE) 8-142
Push Device Subscription (GET) 8-143
Register Application with Push Service (POST) 8-145
Remove Subscription (DELETE) 8-147
Retrieve Device Registration (GET) 8-148
Retrieve Tag (GET) 8-150
Retrieve Web Resource (GET) 8-152
Retrieve Subscription to Push Service. (GET) 8-154
Runtime Configuration (GET) 8-155
Runtime configuration (PUT) 8-157
Runtime (GET) 8-161
Runtime (DELETE) 8-166
Runtime Lock (GET) 8-167
Runtime Lock (DELETE) 8-168
Runtimes (GET) 8-170
Send Bulk Messages (POST) 8-172
Send Message (POST) 8-177
Transaction (GET) 8-182
Transactions (GET) 8-185
Remove Subscription (DELETE) 8-188
Update Device Registration (PUT) 8-190
Update APNs settings (PUT) 8-191
Update GCM settings (PUT). 8-193
Update WNS Settings (PUT). 8-194
Update Tag Information (PUT) 8-196
REST API for the MobileFirst Server push service 8-197
Push Device Registration (DELETE) 8-197
Push Device Registration (GET) 8-198
Push Device Registration (POST) 8-200
Push Device Registrations (GET) 8-202
Push Device Registration (PUT) 8-205
Push Device Subscription (DELETE) 8-207
Push Device Subscription (GET) 8-208
Push Device Subscription (POST) 8-210
Push Device Subscriptions (GET) 8-212
Push Tags (GET). 8-214
Push Applications (GET) 8-217
Push Application (POST) 8-218
Push Application (GET) 8-220
Push Application (DELETE) 8-221
Push Application Settings (GET) 8-222
Push APNS Settings (GET) 8-223
Push APNS settings (PUT) 8-225
Push APNS settings (DELETE) 8-226
Push GCM Settings (GET) 8-227
Push GCM Settings (PUT) 8-228
Push GCM Settings (DELETE) 8-230
Push WNS Settings (GET) 8-231
Push WNS Settings (PUT) 8-232
Push WNS settings (DELETE) 8-234
Push Application (PUT) 8-235
Push Message (POST) 8-236
Push Message (GET) 8-243
Push Message (DELETE) 8-245
Push SMS Settings (GET) 8-245
Push SMS Settings (PUT) 8-247
Push SMS settings (DELETE) 8-250
Push Tags (GET). 8-250
Push Tag (POST) 8-253
Push Tag (GET) 8-255
Push Tag (PUT) 8-256
Push Tag (DELETE) 8-258
Push Webhooks (POST) 8-259
Push Webhooks (PUT). 8-261
Push Webhook (DELETE). 8-263
Push Health Checker (GET) 8-264
Bulk Push Messages (POST) 8-264
REST API for the MobileFirst runtime 8-270
REST API for MobileFirst Analytics and Logger 8-270

Deploying MobileFirst Server to the cloud 9-1

Deploying to the cloud 9-1
IBM MobileFirst Platform Foundation on cloud 9-1
Deploying MobileFirst Server on IBM PureApplication System 9-44
Installing IBM MobileFirst Platform Foundation System Pattern. 9-47
Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern 9-48
Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server. 9-49

Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server 9-54
Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server. 9-59
Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server 9-64
Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers 9-70
Deploying MobileFirst Application Center on a single-node WebSphere Application Server Liberty profile server 9-79
Deploying MobileFirst Application Center on a single-node WebSphere Application Server full profile server 9-82
Configuring MobileFirst administration security with an external LDAP repository 9-86
Configuring an external database with a IBM MobileFirst Platform Foundation System Pattern 9-91
Deploying and configuring MobileFirst Analytics 9-92
Predefined templates for MobileFirst Platform Pattern 9-98
Script packages for MobileFirst Server . . . 9-108
Upgrading IBM MobileFirst Platform Foundation System Pattern 9-126

Administering MobileFirst applications. 10-1

Deploying MobileFirst applications to test and production environments 10-2
Deploying or updating an adapter to a production environment 10-2
Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates 10-3
Building an application for a test or production environment 10-4
Registering an application to a production environment 10-6
Transferring server-side artifacts to a test or production server. 10-7
Updating MobileFirst apps in production 10-14
Administering MobileFirst applications through the MobileFirst Operations Console 10-15
Mobile-application management 10-15
Application status and token licensing . . . 10-20
Error log of operations on runtime environments. 10-20
Audit log of administration operations . . . 10-21
Administering MobileFirst applications through Ant 10-23
Calling the mfpadm Ant task 10-24
Commands for general configuration 10-27
Commands for adapters 10-30
Commands for apps 10-34
Commands for devices 10-41
Commands for troubleshooting. 10-44
Administering MobileFirst applications through the command line 10-47
Calling the mfpadm program 10-48
Commands for general configuration 10-54
Commands for adapters 10-57
Commands for apps 10-61
Commands for devices 10-68
Commands for troubleshooting. 10-70
Federal standards support in IBM MobileFirst Platform Foundation 10-74
FDCC and USGCB support 10-75
FIPS 140-2 support 10-75
License tracking 10-80
Setting the application license information 10-80
License Tracking report 10-81
Token license validation 10-83
Integration with IBM License Metric Tool 10-84

Analytics and Logger. 11-1

Major features 11-1
MobileFirst Analytics Server installation guide 11-2
System requirements. 11-2
Capacity considerations. 11-3
Installing MobileFirst Analytics on WebSphere Application Server Liberty 11-4
Installing MobileFirst Analytics on Tomcat . . 11-6
Installing MobileFirst Analytics on WebSphere Application Server 11-7
Installing MobileFirst Analytics with Ant tasks 11-10
Installing MobileFirst Analytics Server V8.0.0 on servers running previous versions 11-12
Configuration guide 11-14
Configuring analytics from the MobileFirst Operations Console. 11-23
Enabling or disabling data collection from the MobileFirst Operations Console. 11-23
Role-based access control 11-23
Setting Log Filters from the MobileFirst Operations Console. 11-24
Custom charts 11-25
Alerts 11-30
Developing the analytics client 11-35
Analytics SDK 11-35
Logger SDK 11-37
Analytics workflows 11-42
App usage analytics 11-42
Crash capture. 11-45
Custom analytics 11-48
Troubleshooting Analytics and Logger 11-49

Integrating with other IBM products 12-1

Application Center 13-1

Concept of Application Center 13-1
Specific platform requirements 13-2
General architecture 13-3
Preliminary information 13-5
Preparations for using the mobile client 13-6

Importing and building the project (Android, iOS, Windows Phone) 13-8
Customizing features (for experts): Android, iOS, Windows Phone 13-9
Microsoft Windows 8: Building the project 13-11
Deploying the mobile client in Application Center 13-12
Push notifications of application updates . . . 13-12
Configuring push notifications for application updates. 13-13
Configuring the Application Center server for connection to Google Cloud Messaging . . . 13-14
Configuring the Application Center server for connection to Apple Push Notification Services 13-16
Building a version of the mobile client that does not depend on the GCM API. 13-18
The Application Center console. 13-18
Starting the Application Center console . . . 13-19
Troubleshooting a corrupted login page (Apache Tomcat) 13-20
Troubleshooting a corrupted login page in Safari browsers 13-21
Application Management. 13-21
Adding a mobile application 13-22
Adding an application from a public app store 13-24
Application properties. 13-27
Editing application properties 13-29
Upgrading a mobile application in MobileFirst Server and the Application Center. 13-31
Downloading an application file 13-35
Viewing application reviews. 13-35
User and group management 13-36
Access control 13-39
Managing access control 13-39
Device Management 13-41
Application enrollment tokens in Windows 8 Universal 13-44
Signing out of the Application Center console 13-45
Command-line tool for uploading or deleting an application 13-45
Using the stand-alone tool to upload an application 13-46
Using the stand-alone tool to delete an application 13-47
Using the stand-alone tool to clear the LDAP cache 13-48
Ant task for uploading or deleting an application 13-49
The mobile client 13-51
Installing the client on an Android mobile device 13-51
Installing the client on an iOS mobile device 13-54
Installing the client on Windows 8 Universal 13-58
The Login view 13-61
Views in the Application Center client . . . 13-62
Installing an application on an Android device 13-66
Installing an application on an iOS device 13-68
Installing an application on a Windows Phone device 13-70
Installing a Windows Store application on a Windows device. 13-72
Installing applications through public app stores 13-77
Removing an installed application. 13-78
Showing details of a specific application version 13-79
Updating an application 13-80
Upgrading the Application Center client automatically. 13-80
Reverting an installed application 13-84
Marking or unmarking a favorite app . . . 13-85
Submitting a review for an installed application 13-85
Viewing reviews. 13-86
Setting logging and tracing for Application Center on the application server 13-87
Enabling logging and tracing in WebSphere Application Server full profile 13-87
Enabling logging and tracing in WebSphere Application Server Liberty 13-87
Enabling logging and tracing in Apache Tomcat 13-88
JNDI properties for controlling trace output 13-88

Troubleshooting 14-1

Glossary 15-1
A 15-1
B 15-2
C 15-2
D 15-4
E 15-4
F 15-4
G 15-5
H 15-5
I 15-5
J 15-5
K 15-6
L 15-6
M 15-7
N 15-7
O 15-8
P 15-8
R 15-9
S 15-9
T 15-10
U 15-11
V 15-11
W 15-11
X 15-11

Support and comments. 16-1

Notices A-1
Trademarks A-3
Terms and conditions for product documentation A-3
IBM Online Privacy Statement A-4

Index X-1

IBM MobileFirst Platform Foundation V8.0.0 documentation

Welcome to the IBM MobileFirst™ Platform Foundation V8.0.0 documentation, where you can find information about how to install, maintain, and use the IBM MobileFirst Platform Foundation.

Getting started

"Product overview" on page 2-1
IBM MobileFirst Platform Foundation 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 and additional resources" on page 4-1
Tutorials help you get started with and learn about IBM MobileFirst Platform Foundation. Use them to evaluate what the product can do for you.

"Installation overview" on page 6-1
IBM MobileFirst Platform Foundation provides development tools and server-side components that you can install on-premises or deploy to the cloud for test or production use. Review the installation topics appropriate for your installation scenario.

"Configuring MobileFirst Server" on page 6-164
Consider your backup and recovery policy, optimize your MobileFirst Server configuration, and apply access restrictions and security options.

"System requirements" on page 2-7
System requirements for IBM MobileFirst Platform Foundation include operating systems, SDKs, and other software.

Common tasks

"Developing applications" on page 7-1
The process for developing applications has steps that are common to all environments: setting up a server, creating an initial server registration and corresponding configuration files, creating a new (or opening an existing) project in your chosen IDE, and adding the necessary SDK files to your IDE project. Also, server-side adapters can be developed as needed for the application.

"Deploying MobileFirst Server to the cloud" on page 9-1
You can deploy MobileFirst Server to the cloud. Review the various options to run MobileFirst Server on the cloud.

"Administering MobileFirst applications" on page 10-1
Run and maintain MobileFirst applications in production.

"Application Center" on page 13-1
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 14-1
You can find advice on how to troubleshoot problems, and more information about known limitations and technotes (Troubleshooting).

"Known issues" on page 3-25
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 features for IBM MobileFirst Platform Foundation" on page 2-10
Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

IBM Support home for IBM MobileFirst Platform Foundation

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 is an integrated platform that helps you extend your business to mobile devices.

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

Product main capabilities

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

Development

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

You can develop apps by using HTML5, CSS3, and JavaScript. You can optionally write native code (Java™ or Objective-C). IBM MobileFirst Platform Foundation 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 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 includes a unified notification framework which provides a consistent mechanism for such push notifications. With this unified notification framework, you can send push notifications without having to know the details of each targeted device or platform because each mobile platform has a different mechanism for push notification.

Offline mode

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

Update

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

Security

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

IBM MobileFirst Platform Foundation provides secure, end-to-end communication by positioning a server that oversees the flow of data between the mobile application and your back-end systems. With IBM MobileFirst Platform Foundation, you can define custom security handlers for any access to this flow of data. Because any access to data of a mobile application has to go through this server instance, you can define different security handlers for mobile applications, web applications, and back-end access. With this kind of granular security, you can define separate levels of authentication for different functions of your mobile application. 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 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 includes a range of operational analytics and reporting mechanisms for collecting, viewing, and analyzing data from your IBM MobileFirst Platform Foundation applications and servers, and for monitoring server health.

Application publishing

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

The concept of the Application Center is similar to the concept of the Apple public App Store or the Google Play store, except that it targets the development process.

The Application Center provides a repository for storing the mobile application files and a web-based console for managing that repository. The Application Center also provides a mobile client application to allow users to browse the catalog of applications that are stored by the Application Center, install applications, leave feedback for the development team, and expose production applications to IBM® Endpoint Manager. Access to download and install applications from the Application Center is controlled by using access control lists (ACLs).

Product components

IBM MobileFirst Platform Foundation consists of the following components: MobileFirst Platform CLI, MobileFirst Server, client-side runtime components, MobileFirst Operations Console, Application Center, and IBM MobileFirst Platform Foundation System Pattern.

Component overview

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

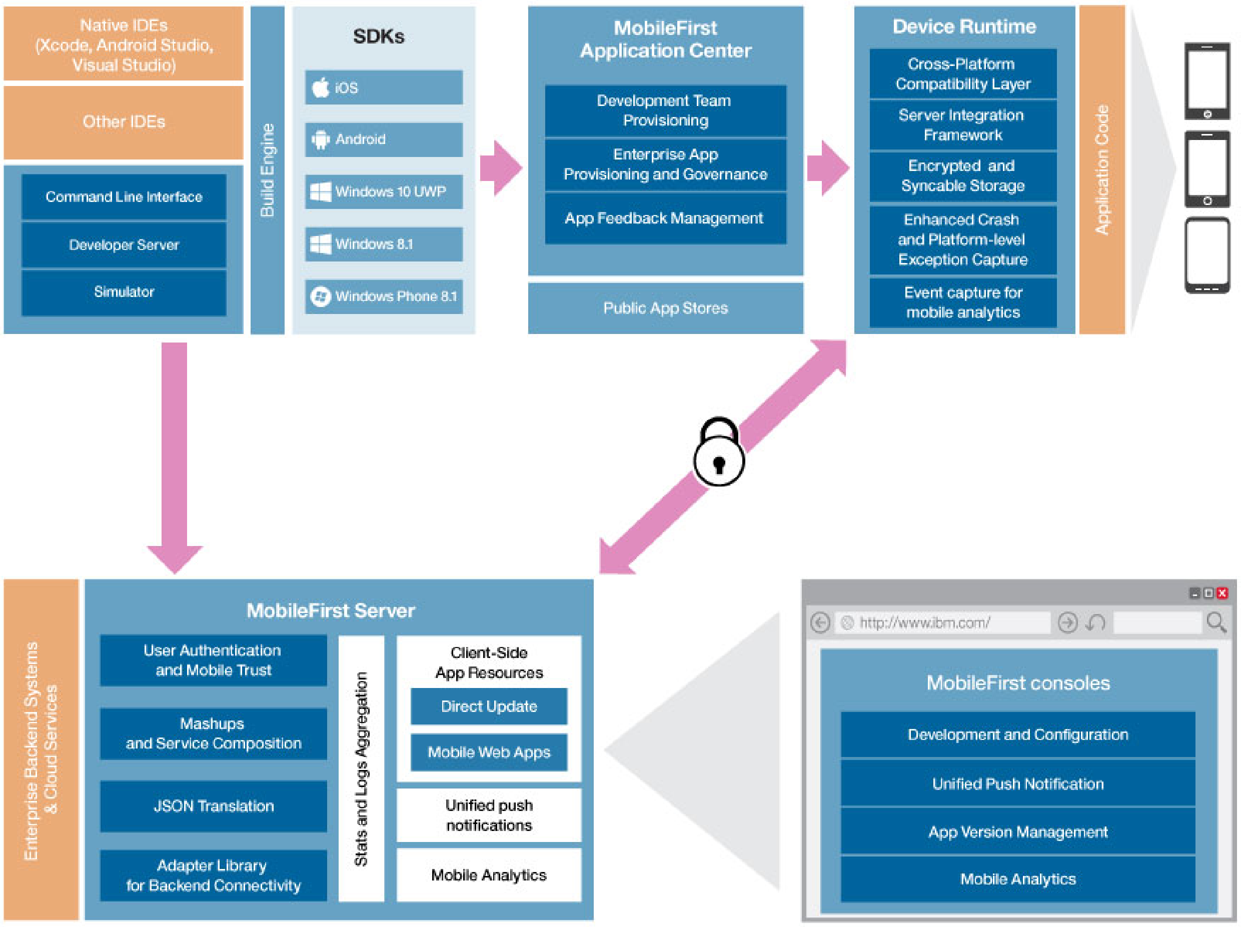

Native IDEs (Xcode, Android Studio, Visual Studio)
Other IDEs
Command Line Interface
Developer Server
Simulator
Build Engine
SDKs
iOS
Android
Windows 10 UWP
Windows 8.1
Windows Phone 8.1
MobileFirst Application Center
Development Team Provisioning
Enterprise App Provisioning and Governance
App Feedback Management
Public App Stores
Device Runtime
Cross-Platform Compatibility Layer
Server Integration Framework
Encrypted and Syncable Storage
Enhanced Crash and Platform-level Exception Capture
Event capture for mobile analytics
Application Code
Enterprise Backend Systems & Cloud Services
MobileFirst Server
User Authentication and Mobile Trust
Mashups and Service Composition
JSON Translation
Adapter Library for Backend Connectivity
Stats and Logs Aggregation
Client-Side App Resources
Direct Update
Mobile Web Apps
Unified push notifications
Mobile Analytics
http://www.ibm.com/
MobileFirst consoles
Development and Configuration
Unified Push Notification
App Version Management
Mobile Analytics

Figure 2-1. IBM MobileFirst Platform Foundation architecture

MobileFirst Platform CLI

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to develop and manage applications, in addition to using the IBM MobileFirst Platform Operations Console. Some aspects of the MobileFirst development process must be done with the CLI.

The commands, all prefaced with **`mfpdev`**, support the following types of tasks:

- Registering apps with the MobileFirst Server
- Configuring your app
- Creating, building, and deploying adapters
- Previewing and updating Cordova apps

For more information, see “The MobileFirst command-line interface (CLI)” on page 7-13

MobileFirst Server

The MobileFirst Server provides secured backend connectivity, application management, push notification support and analytics capabilities and monitoring to MobileFirst applications. 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 application packages, and is in fact a collection of web applications, optionally packaged as an EAR (enterprise archive) file that 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.

- For more information about using the MobileFirst Server in a development environment, see “Setting up the MobileFirst Development Server” on page 7-12
- For more information about installing the MobileFirst Server on-premises, see “Installing IBM MobileFirst Platform Server” on page 6-2
- For more information about deploying the MobileFirst Server to the cloud, see “Deploying MobileFirst Server to the cloud” on page 9-1

Client-side runtime components

IBM MobileFirst Platform Foundation provides client-side runtime code that embeds server functionality within the target environment of deployed apps. These runtime client APIs are libraries that are integrated into the locally stored app code. You use them to add MobileFirst features to your client apps. The APIs and libraries can be installed with the IBM MobileFirst Platform Foundation Developer Kit or you can download them from repositories for your development platform.

- For more information about iOS SDKs, see “Developing native applications for iOS in Xcode” on page 7-27
- For more information about Android SDKs, see “Developing native applications in Android Studio” on page 7-52
- For more information about Windows SDKs, see “Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal” on page 7-65
- For more information about Cordova plug-ins, see “Developing Cordova applications” on page 7-83

MobileFirst Operations Console

The MobileFirst Operations Console is used for the control and management of the mobile applications. The MobileFirst Operations Console is also an entry point to learn about IBM MobileFirst Platform development. From the console, you can download code examples, tools, and SDKs.

You can use the MobileFirst Operations Console for the following tasks:

- Monitor and configure all deployed applications, adapters, and push notification rules from a centralized, web-based console.
- 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, pre-configured reports about user adoption and usage (number and frequency of users that are engaging with the server through the applications).
- Configure data collection rules for application-specific events.

For more information about using the console for development, see “MobileFirst Operations Console overview” on page 7-6.

For more information about using the console for application management, see “Administering MobileFirst applications” on page 10-1

IBM MobileFirst Analytics

IBM MobileFirst Platform Foundation 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 and Android devices (crash events for native code and JavaScript 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.

For more information about IBM MobileFirst Analytics, see “Analytics and Logger” on page 11-1.

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.

For more information, see "Application Center" on page 13-1

IBM MobileFirst Platform Foundation System Pattern

With the MobileFirst Platform Pattern, you can deploy MobileFirst Server on IBM PureApplication® System or IBM PureApplication Service on SoftLayer®. With these patterns, administrators and corporations can respond quickly to changes in the business environment by taking advantage of on-premises Cloud technologies. This approach simplifies the deployment process, and improves the operational efficiency to cope with increased mobile demand. The demand accelerates iteration of solutions that exceed traditional demand cycles. Using MobileFirst Platform Pattern also gives access to best practices and built-in expertise, such as built-in scaling policies.

PureApplication System

IBM PureApplication System is an integrated, highly scalable system that is based on IBM X-Architecture, providing an application-centric computing model in a cloud environment.

An application-centric system is an efficient way to manage complex applications and the tasks and processes that are invoked by the application. The entire system implements a diverse virtual computing environment, in which different resource configurations are automatically tailored to different application workloads. The application management capabilities of the IBM PureApplication System platform make deployment of middleware and other application components quick, easy, and repeatable.

IBM PureApplication System provides virtualized workloads and a scalable infrastructure that is delivered in one integrated system.

Virtual System Patterns

Virtual system patterns are a logical representation of a recurring topology for a set of deployment requirements.

Virtual system patterns enable efficient and repeatable deployments of systems that include one or more virtual machine instances, and the applications that run on them. You can completely automate the deployment and eliminate the need to perform multiple time-consuming manual tasks. Such a deployment eliminates problems that are introduced by error-prone, manual configuration processes, especially in complex production topologies such as server farms, and accelerates solution deployment.

System requirements

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

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

System requirements by type (high-level)

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

- Operating systems
- Software

System requirements by platform (detail)

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

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

System requirements by component (detail)

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

- IBM MobileFirst Platform Command Line Interface (CLI)
- IBM Mobile Foundation for Bluemix®
- IBM MobileFirst Platform Application Center
- IBM MobileFirst Platform Operational Analytics
- IBM MobileFirst Platform Server
- IBM MobileFirst Platform Device Runtime

Licensing in MobileFirst Server

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

Application or Addressable Device licenses

If you have purchased Application or Addressable Device 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.

Processor value unit (PVU) licensing

Processor value unit (PVU) licensing is available if you have purchased IBM MobileFirst Platform Foundation Extension (see http://www.ibm.com/software/sla/sladb.nsf/lilookup/C154C7B1C8C840F38525800A0037B46E?OpenDocument), but only after the purchase of IBM WebSphere® Application Server Network Deployment, IBM API Connect™ Professional, or IBM API Connect Enterprise.

The PVU license pricing structure is responsive to both the type and number of processors that are available to installed products. Entitlements can be full capacity or subcapacity. Under the processor value unit licensing structure, you license software based on the number of value units assigned to each processor core.

For example, processor type A is assigned 80 value units per core and processor type B is assigned 100 value units per core. If you license a product to run on two type A processors, you must acquire an entitlement for 160 value units per core. If the product is to run on two type B processors, the required entitlement is 200 value units per core.

For more information on PVU licensing see https://www.ibm.com/support/knowledgecenter/SS8JFY_9.2.0/com.ibm.lmt.doc/Inventory/overview/c_processor_value_unit_licenses.html.

Token Licensing

If you have purchased *Token licenses*, configure your MobileFirst Server to communicate with a remote token license server.

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 10-83.

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 6-150.

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 “Setting the application license information” on page 10-80.

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. 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 6-150.

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

Downloading IBM MobileFirst Platform Foundation V8.0.0

The first step for you to work with IBM MobileFirst Platform Foundation V8.0.0 is to download the artifacts that are required for its installation.

You can download the artifacts of IBM MobileFirst Platform Foundation V8.0.0 in multiple ways, depending on how you purchased this product.

For detailed instructions on how to download the product artifacts that you need for its installation, see the product Download page.

Matrix of features and platforms

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

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

Accessibility features for IBM MobileFirst Platform Foundation

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

Accessibility features

IBM MobileFirst Platform Foundation includes the following major accessibility features:

- Keyboard-only operation
- Operations that support the use of a screen reader

IBM MobileFirst Platform Foundation uses the latest W3C Standard, WAI-ARIA 1.0 (http://www.w3.org/TR/wai-aria/), to ensure compliance to US Section 508 (http://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards), and Web Content Accessibility Guidelines (WCAG) 2.0 (http://www.w3.org/TR/WCAG20/). To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest web browser that is supported by this product.

The IBM MobileFirst Platform Foundation online product documentation in IBM Knowledge Center is enabled for accessibility. The accessibility features of IBM Knowledge Center are described at: http://www.ibm.com/support/knowledgecenter/doc/kc_help.html#accessibility.

Keyboard navigation

This product uses standard navigation keys.

Interface information

The IBM MobileFirst Platform Foundation user interfaces do not have content that flashes 2 - 55 times per second.

You can use a screen reader with a digital speech synthesizer to hear what is displayed on your screen. Consult the documentation with your assistive technology for details about how to use it with this product and its documentation.

MobileFirst Platform CLI
By default, status messages that are displayed by the MobileFirst Platform CLI use various colors to indicate success, errors, and warnings. You can

use the --no-color option on any MobileFirst Platform CLI 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.

Web interface

The IBM MobileFirst Platform Foundation web user interfaces rely on cascading style sheets to render content properly and to provide a usable experience. The application provides an equivalent way for low-vision users to use a user's system display settings, including high-contrast mode. You can control font size by using the device or web browser settings.

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 Analytics Console, the IBM MobileFirst Platform Application Center console, and the IBM MobileFirst Platform Application Center mobile client.

The IBM MobileFirst Platform Foundation web user interface includes WAI-ARIA navigational landmarks that you can use to quickly navigate to functional areas in the application.

Installation and configuration

There are two ways to install and configure IBM MobileFirst Platform Foundation: 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. You can read about the accessibility features of IBM Installation Manager in the IBM Knowledge Center.

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 6-48

  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 6-204. In this case, the installation and the upgrading of Application Center can be done independently.
- "Installing with Ant Tasks" on page 6-110
- "Installing the Application Center with Ant tasks" on page 6-204

Vendor software

IBM MobileFirst Platform Foundation includes certain vendor software that is not covered under the IBM license agreement. IBM makes no representation about the accessibility features of these products. Contact the vendor for the accessibility information about its products.

Related accessibility information

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service
800-IBM-3383 (800-426-3383)
(within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see IBM Accessibility (www.ibm.com/able).

Release notes

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

What's new in V8.0.0

IBM MobileFirst Platform Foundation V8.0.0 brings significant changes that modernize your MobileFirst application development, deployment, and management experience.

What's new in building apps

The IBM MobileFirst Platform Foundation SDK and command-line interface have been redesigned to give you greater flexibility and efficiency when developing your apps. Also, you can now use any of your preferred Cordova tools when you develop cross-platform apps.

Review the following sections to learn what is new for developing your apps.

New development and deployment process

Starting with IBM MobileFirst Platform Foundation V8.0.0, you no longer create a project WAR file that needs to be installed in the application server. Instead, the MobileFirst Server is installed once, and you upload the server-side configuration of your apps, of the resource security or of the push service to the server. You can modify the configuration of your apps with the MobileFirst Operations Console.

MobileFirst projects no longer exist. Instead, you develop your mobile app with the development environment of your choice.

You can modify the server-side configuration of your apps and adapters without stopping the MobileFirst Server.

For more information about the new development process, see “Development concepts and overview” on page 7-2

For more information about the migration of existing applications, see “Migrating apps from earlier releases” on page 5-1.

For more information about administering MobileFirst applications, see “Administering MobileFirst applications” on page 10-1.

Web applications

You can now use the MobileFirst client-side JavaScript API to develop web applications with your preferred tools and IDE. You can register your web application to MobileFirst Server to add security capabilities to the application. See What's new in web-applications security.

You can also use the new client-side JavaScript web analytics API, which is provided as part of the new web SDK, to add IBM MobileFirst Analytics capabilities to your web application.

For more information about developing MobileFirst web applications, see "Developing web applications" on page 7-73.

Develop cross-platform apps with your preferred Cordova tools

Starting with IBM MobileFirst Platform Foundation V8.0.0, you can now use your preferred Cordova tools (such as Apache Cordova CLI or Ionic Framework) to develop your cross-platform hybrid apps. You obtain these tools independently of IBM MobileFirst Platform Foundation, and then add MobileFirst plug-ins to provide MobileFirst back-end capabilities. For more information, see "Developing Cordova applications" on page 7-83.

You can also take advantage of additional new security features for Cordova apps.

You can install the IBM MobileFirst Platform Foundation Studio Eclipse plug-in to manage your cross-platform Cordova apps that are enabled with IBM MobileFirst Platform Foundation in the Eclipse development environment. The IBM MobileFirst Platform Foundation Studio plug-in also provides additional IBM MobileFirst Platform Command Line Interface (CLI) commands that you can run from within the Eclipse environment. For more information, see "IBM MobileFirst Studio plug-in for managing Cordova projects in Eclipse" on page 7-114.

SDK componentization

Previously MobileFirst client SDK was delivered as a single framework or JAR file. You can now choose to include or exclude specific functionalities. In addtion to the core SDK, each MobileFirst API has its own set of optional components. See "Adding optional iOS frameworks" on page 7-31, "Adding the optional MobileFirst components with Gradle" on page 7-56, "Adding MobileFirst features to an existing Cordova app" on page 7-91 and "Adding the optional MobileFirst components by using NuGet" on page 7-67.

New, improved development command-line interface (CLI)

The IBM MobileFirst Platform Command Line Interface (CLI) has been redesigned for greater development efficiency, including for use in automated scripts. Commands now start with the prefix **`mfpdev`**. The CLI is included in the IBM MobileFirst Platform Foundation Developer Kit, or you can quickly download the latest version of the CLI from npm. For more information, see "The MobileFirst command-line interface (CLI)" on page 7-13.

Migration assistance tool

A migration assistance tool simplifies the procedure of migrating your existing apps to IBM MobileFirst Platform Foundation version 8.0. The tool scans your existing MobileFirst apps and creates a list of the APIs that are used in the file that are either removed, deprecated, or replaced in version 8.0. When you run the migration assistance tool on Apache Cordova applications that were created with the IBM MobileFirst Platform Foundation, it creates a new Cordova structure for the app that is compliant with version 8.0. For more information about the migration assistance tool, see "Scanning existing MobileFirst native iOS apps to prepare for MobileFirst version 8.0" on page 5-17, "Scanning existing MobileFirst native Android apps to prepare for MobileFirst version 8.0" on page 5-25, "Scanning existing MobileFirst native Windows apps to prepare for MobileFirst version 8.0" on page 5-30, "Starting the Cordova app migration with the migration assistance tool" on page 5-38.

Cordova Crosswalk WebView

Starting with Cordova 4.0 the pluggable WebView allows the default web runtime to be replaced. Crosswalk is now supported by Cordova applications with IBM MobileFirst Platform Foundation. Using the Crosswalk WebView for Android allows high performance and consistent user experience across a wide range of mobile devices. To take advantage of the Crosswalk capabilities, apply the Cordova Crosswalk plug-in. For more information see “Crosswalk WebView (Android)” on page 7-133.

Distributing MobileFirst SDK for Windows 8 and Windows 10 Universal apps with NuGet

The MobileFirst SDK for Windows 8 and Windows 10 Universal apps is available from NuGet at https://www.nuget.org/packages. To get started, see the following topics:
- “Adding the MobileFirst SDK by using NuGet” on page 7-66
- “Methods of setting up your environment” on page 7-65

org.apache.http replaced by okHttp

`org.apache.http` has been removed from the Android SDK. `okHttp` will be used as the http dependency. For details on the SDK changes see org.apache.http removal.

Application Samples

IBM MobileFirst Platform Foundation now includes sample applications that you can use as a starting point for developing your own apps. Samples for iOS, Android, Windows, web, and Cordova are provided. You can download the sample apps from the MobileFirst Operations Console. For more information, see “Getting started with a sample MobileFirst application” on page 7-25.

WKWebView support for iOS hybrid Cordova apps

From IBM MobileFirst Platform Foundation V8.0.0, you can replace the default UIWebView in cordova apps with WKWebView. To get started with WKWebView for your hybrid iOS cordova applications, refer “WKWebView (iOS)” on page 7-133.

What's new in MobileFirst APIs

New features improve and extend the APIs that you can use to develop mobile applications. Use the latest APIs to take advantage of new, improved, or changed functions in IBM MobileFirst Platform Foundation.

In addition to the MobileFirst APIs that were updated and improved, some APIs are deprecated or discontinued.
- “Deprecated features and API elements” on page 3-17
- “Discontinued features and API elements” on page 3-19

Updated JavaScript server-side API

In V8.0.0, back-end invocation functions are supported only for adapter types that are supported. Currently, only HTTP and SQL adapters are supported, so back-end invokers WL.Server.invokeHttp and WL.Server.invokeSQL are supported, too.

New Java server-side API

IBM MobileFirst Platform Foundation V8.0.0 provides new Java server-side API, which you can use to extend MobileFirst Server.

New Java server-side API for security
The new security API package, com.ibm.mfp.server.security.external, and its contained packages, include the interfaces that are required for developing security checks and adapters that use the security-check context. See "Java server-side API" on page 8-7.

New Java server-side API for client registration data
The new client registration-data API package, com.ibm.mfp.server.registration.external, and its contained packages, include an interface for providing access to persistent MobileFirst client registration data. See "Java server-side API" on page 8-7.

Application getJaxRsApplication()

With this new API, you can return the JAX-RS application for the adapter.

String getPropertyValue (String propertyName)

With this new API, you can get the value from the adapter configuration (or default value).

For more information, see "MobileFirst server-side API" on page 8-6.

Updated Java server-side API

IBM MobileFirst Platform Foundation V8.0.0 also includes updated Java server-side API, which you can use to extend MobileFirst Server.

getMFPConfigurationProperty(String name)

The signature of this new API has not changed in this version. However, its behavior is now identical to that of String getPropertyValue (String propertyName), which is described in "New Java server-side API."

WLServerAPIProvider
In V7.0.0 and V7.1.0, the Java API was accessible through the WLServerAPIProvider interface. For example:

```
WLServerAPIProvider.getWLServerAPI.getConfigurationAPI();
```

and

```
WLServerAPIProvider.getWLServerAPI.getSecurityAPI();
```

These static interfaces are still supported in V8.0.0, to allow adapters that were developed in previous versions of the product to compile and deploy. Old adapters that do **not** use push notifications or the previous security API continue to work in V8.0.0. Adapters that **do** use push notifications or the previous security API break.

For more information, see "MobileFirst server-side API" on page 8-6.

JavaScript client-side APIs for web applications

The JavaScript client-side API that is used for development of cross-platform Cordova applications is now available also for development of web applications, with slight variations in the initialization method. Note that not all functions of the JavaScript API are applicable to web applications. For a full API reference, see the JavaScript client-side API reference.

In addition, a new JavaScript client-side web analytics API is provided for adding IBM MobileFirst Analytics capabilities to your web application. For a full API reference, see "JavaScript web analytics client-side API" on page 8-4.

For more information about using these APIs to develop MobileFirst web applications, see "Developing web applications" on page 7-73.

Updated C# client-side API for Windows 8 Universal and Windows Phone 8 Universal

The C# client-side API for Windows 8 Universal and Windows Phone 8 Universal have changed. For more information, see "C# client-side API for Windows 10 Universal Windows Platform and Windows 8 Universal apps" on page 8-6.

New Java client-side APIs for Android

`public void getDeviceDisplayName(final DeviceDisplayNameListener listener);`

`public void setDeviceDisplayName(String deviceDisplayName,final WLRequestListener listener);`
: With this new method, you can set the display name of a device in the MobileFirst Server registration data.

New Objective-C client-side APIs for iOS

`(void) getDeviceDisplayNameWithCompletionHandler:(void(^)(NSString *deviceDisplayName , NSError *error))completionHandler;`
: With this new method, you can get the display name of a device from the MobileFirst Server registration data.

`(void) setDeviceDisplayName:(NSString*)deviceDisplayName WithCompletionHandler:(void(^)(NSError* error))completionHandler;`
: With this new method, you can set the display name of a device in the MobileFirst Server registration data.

New push client-side APIs

Push client-side API is supported with IBM MobileFirst Platform Foundation V8.0.0.

To use push API for Android apps, see "Java client-side push API for Android apps" on page 8-6.

To use push API for iOS apps, see "Objective-C client-side push API for iOS apps" on page 8-5.

Updated REST API for the administration service

The REST API for the administration service is partly refactored. In particular, the API for beacons and mediators is removed and most REST services for push notification are now part of the REST API for the push service. For more information, see "REST API for the MobileFirst Server administration service" on page 8-7 and "REST API for the MobileFirst Server push service" on page 8-197.

Updated REST API for the runtime

The REST API for the MobileFirst runtime now provides several services for mobile clients and confidential clients to call adapters, obtain access tokens, get Direct Update content, and more. Most of the REST API endpoints are protected by OAuth. On a development server, you can view the Swagger doc for the runtime API at:

`http(s)://<server_ip>:<server_port>/<context_root>/doc`

For more information, see "REST API for the MobileFirst runtime" on page 8-270.

What's new in MobileFirst security

The security framework in IBM MobileFirst Platform Foundation was entirely redesigned. New security features were introduced, and some modifications were made to existing features.

Security framework overhaul

The MobileFirst security framework was redesigned and reimplemented to improve and simplify security development and administration tasks. The framework is now inherently based on the OAuth model, and the implementation is session-independent. See "Overview of the MobileFirst security framework" on page 7-265.

On the server side, the multiple building blocks of the framework were replaced with security checks (implemented in adapters), allowing for simplified development with new APIs. Sample implementations and predefined security checks are provided. See "Security checks" on page 7-281. Security checks can be configured in the adapter descriptor, and customized by making runtime adapter or application configuration changes, without redeploying the adapter or disrupting the flow. The configurations can be done from the redesigned MobileFirst Operations Console security interfaces. You can also edit the configuration files manually, or use the MobileFirst Platform CLI or **mfpadm** tools. See "Security-checks configuration" on page 7-297.

See the other security release notes for specific changes and additions that are also the result of the security-framework redesign.

Application-authenticity security check

MobileFirst application-authenticity validation is now implemented as a predefined security check that replaces the previous "extended application authenticity checking". You can dynamically enable, disable, and configure application-authenticity validation by using either MobileFirst Operations Console or **mfpadm**. A stand-alone MobileFirst application-authenticity Java tool (`mfp-app-authenticity-tool.jar`) is provided for generating an application-authenticity file. See "Application-authenticity security check" on page 7-282.

Confidential clients

The support for confidential clients was redesigned and reimplemented using the new OAuth security framework. See "Confidential clients" on page 7-279.

Web-applications security

The revised OAuth-based security framework supports web applications. You can now register web applications with MobileFirst Server to add security capabilities to your application and protect access to your web resources. For more information about developing MobileFirst web applications, see "Developing web applications" on page 7-73. The application-authenticity security check is not supported for web applications.

Cross-platform applications (Cordova apps), new and changed security features

Additional security features are available to help protect your Cordova app. These features include the following:

- Web resources encryption: Use this feature to encrypt the web resources in your Cordova package to help prevent someone from modifying the package. For more information, see "Encrypting the web resources of your Cordova packages" on page 7-112.
- Web resources checksum: Use this feature to run a checksum test that compares the current statistics of the web resources of the app with the baseline statistics that were established when it was first opened. This check helps to prevent someone from modifying the app after it is installed and opened. For more information, see "Enabling the web resources checksum feature" on page 7-113.
- Certificate pinning: Use this feature to associate the certificate of an app with a certificate on the host server. This feature helps to prevent information that is passed between the app and the server from being viewed or modified. For more information, see "Certificate pinning" on page 7-185.
- Support for the Federal Information Processing Standard (FIPS) 140-2: Use this feature to ensure that data that is transferred is compliant with the FIPS 140-2 cryptography standard. For more information, see "Enabling FIPS 140-2" on page 10-77.
- OpenSSL: To use OpenSSL data encryption and decryption with your Cordova app for the iOS platform, you can use the `cordova-plugin-mfp-encrypt-utils` Cordova plug-in. For more information, see "Cordova plug-ins for MobileFirst features" on page 7-87 and "Enabling OpenSSL for Cordova iOS" on page 7-126.

Device Single Sign-On (SSO)

Device single sign-on (SSO) is now supported by way of the new predefined **`enableSSO`** security-check application-descriptor configuration property. See "Configuring device single sign-on (SSO)" on page 7-301.

Direct Update

In contrast to earlier versions of MobileFirst, starting with V8.0.0:

- If a client application accesses an unprotected resource, the application does not receive updates, even if an update is available on MobileFirst Server. See "Updating Cordova client apps directly" on page 7-235.
- After it has been activated, Direct Update is enforced on every request for a protected resource.

External-resources Protection

The supported method and provided artifacts for protecting resources on external servers were modified:

- A new, configurable MobileFirst Java Token Validator access-token validation module is provided for using the MobileFirst security framework to protect resources on any external Java server. The module is provided as a Java library (`mfp-java-token-validator-8.0.0.jar`), and replaces the use of the obsolete MobileFirst Server token-validation endpoint to create a custom Java validation module. See "MobileFirst Java Token Validator" on page 7-274.
- The MobileFirst OAuth Trust Association Interceptor (TAI) filter, for protecting Java resources on an external WebSphere Application Server or WebSphere Application Server Liberty server, is now provided as a Java library (`com.ibm.imf.oauth.common_8.0.0.jar`). The library uses the new Java Token Validator validation module, and the configuration of the provided TAI changed. See "MobileFirst OAuth Trust Association Interceptor (TAI) for protecting resources on WebSphere Java servers" on page 7-275.
  The server-side MobileFirst OAuth TAI API is no longer required and was removed.
- The `passport-mfp-token-validation` MobileFirst Node.js framework, for protecting Java resources on an external Node.js server, was modified to support the new security framework. See "MobileFirst Node.js resource protection" on page 7-275.
- You can also write your own custom filter and validation module, for any type of resource server, which uses the new introspection endpoint of the authorization server. See "External resources protection" on page 7-274.

Integration with WebSphere DataPower® as an authorization server

You can now select to use WebSphere DataPower as the OAuth authorization server, instead of the default MobileFirst Server authorization server. You can configure DataPower to integrate with the MobileFirst security framework. See "Configuring IBM WebSphere DataPower as the OAuth authorization server" on page 7-314.

LTPA-based single sign-on (SSO) security check

Support for sharing user authentication among servers that use WebSphere light-weight third-party authentication (LTPA) is now provided by using the new predefined LTPA-based single sign-on (SSO) security check. This check replaces the obsolete MobileFirst LTPA realm, and eliminates the previous required configuration. See "LTPA-based single sign-on (SSO) security check" on page 7-285.

Mobile-application management with MobileFirst Operations Console

Some changes were made to the support for tracking and managing mobile applications, users, and devices from IBM MobileFirst Platform Operations Console.
Blocking device or application access is applicable only to attempts to access protected resources.
See "Mobile-application management" on page 10-15.

MobileFirst Server keystore

A single MobileFirst Server keystore is used for signing OAuth tokens and Direct Update packages, and for mutual HTTPS (SSL) authentication. You can dynamically configure this keystore by using either MobileFirst Operations Console or **`mfpadm`**. See “Configuring the MobileFirst Server keystore” on page 7-316.

Native encryption and decryption for iOS

OpenSSL has been removed from the main framework for iOS and replaced by a native encryption/decryption. OpenSSL can be added as a separate framework. See “Enabling OpenSSL for iOS” on page 7-47. For iOS Cordova JavaScript, OpenSSL is still embedded in the main framework. For both APIs, both native and OpenSSL encryption are available.

What's new in operating system support

IBM MobileFirst Platform Foundation V8.0.0 introduces support for Windows 10 Universal apps as well as Apple watchOS 2.

Support for universal applications for Windows 10 Native

With IBM MobileFirst Platform Foundation, you can now write native C# Universal App Platform applications to use the MobileFirst SDK within your app.

To get started with writing your native C# Universal App Platform applications for the MobileFirst SDK, see the following topics:

- “Developing MobileFirst applications” on page 7-24
- “Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal” on page 7-65
- “Client property file for Windows 10 Universal Windows Platform and Windows 8 Universal” on page 7-72

Windows 10 UWP support for Windows hybrid environments

Windows 10 Universal Windows Platform (UWP) support for Windows hybrid environments. For more information on how to get started, see “Developing Cordova apps for Windows” on page 7-128.

BlackBerry end of support

The BlackBerry environment is no longer supported in IBM MobileFirst Platform Foundation.

Bitcode

Bitcode builds are now supported for iOS projects. However, the MobileFirst application-authenticity security check is not supported for apps built with bitcode. For more information, see “Working with bitcode in iOS apps” on page 7-48.

Apple watchOS 2

Apple watchOS 2 is now supported and requires bitcode builds. See “Developing for watchOS 2” on page 7-48.

What's new in deploying and managing apps

IBM MobileFirst Platform FoundationV8.0.0 comes with capabilities to help you deploy and manage your apps. You can now update your apps and adapters without restarting MobileFirst Server.

Improved DevOps support

MobileFirst Server has been significantly redesigned to better support your DevOps environment. MobileFirst Server is installed once into your application server environment, and no changes to the application server configuration are required when you upload an application or change the MobileFirst Server configuration.

You do not need to restart MobileFirst Server when you update your apps or any adapters that your apps depend on. You can perform configuration operations, or upload a new version of an adapter or register a new application while the server is still handling traffic.

Configuration changes and development operations are protected by security roles. For details, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

You can upload development artifacts to the server in various ways to give you more operational flexibility:

- MobileFirst Operations Console is enhanced: In particular, you can now use it to register an application or a new version of an application, to manage app security parameters, and to deploy certificates, create push notification tags, and send push notifications. The console now also includes contextual help guides. For details, see "MobileFirst Operations Console overview" on page 7-6.
- Command-line tool
- For details, see "Administering MobileFirst applications through the command line" on page 10-47.
- REST API calls: For details, see "REST API for the MobileFirst Server administration service" on page 8-7.

Development artifacts that you upload to the server include adapters and their configuration, security configurations for your apps, push notification certificates, and log filters.

For more information about the new MobileFirst Server design, see "Deploying MobileFirst applications to test and production environments" on page 10-2 and "Development concepts and overview" on page 7-2.

Running applications that were created on IBM Bluemix on IBM MobileFirst Platform Foundation

Developers can migrate IBM Bluemix applications to run on IBM MobileFirst Platform Foundation. Migration requires that you make configuration changes to your client application to match IBM MobileFirst Platform Foundation APIs.

IBM MobileFirst Platform Foundation as a service on IBM Bluemix

You can now use the IBM Mobile Foundation for Bluemix service on IBM Bluemix to create and run your enterprise mobile apps. For more information, see

Deploying the MobileFirst Server to the cloud.

No `.wlapp` files

In previous versions, applications were deployed to MobileFirst Server by uploading a `.wlapp` file. The file contained data that described the application and, in the case of hybrid applications, the required web resources also. In V8.0.0, instead of the `.wlapp` file:

- You register an app in MobileFirst Server by deploying an application descriptor JSON file. For more information, see "Developing applications" on page 7-1.
- To update Cordova applications by using Direct Update, you upload an archive (.zip file) of the modified web resource to the server. The archive file no longer contains the web preview files or skins that were possible in previous versions of IBM MobileFirst Platform Foundation. These have been discontinued. The archive contains only the web resources that are sent to the clients, as well as checksums for Direct Update validations. For more information, see Updating Cordova client apps directly.

To enable Direct Update of client Cordova apps that are installed on end-user devices, you must now deploy the modified web resources as an archive (`.zip` file) to the server. To enable *secure* Direct Update, a user-defined keystore file must be deployed in MobileFirst Server and a copy of the matching public key must be included in the deployed client application. For more information about Direct Update, see "Updating Cordova client apps directly" on page 7-235.

Adapters

Adapters are Apache Maven projects

Starting from V8.0.0, MobileFirst adapters are treated as Maven projects. You can create, build, and deploy adapters by using standard command-line Maven commands or using any IDE that supports Maven, such as Eclipse and IntelliJ. For more information, see "Adapters as Apache Maven projects" on page 7-189.

Adapter configuration and deployment in DevOps environments

- Starting with V8.0.0 of MobileFirst Server, administrators can use the MobileFirst Operations Console to modify the behavior of an adapter that has been deployed. After reconfiguration, the changes take effect in the server immediately, without the need to redeploy the adapter, or restart the server. For more information see "Configuring adapters" on page 7-227.
- Starting with V8.0.0, you can "hot deploy" adapters, meaning deploy, undeploy, and redeploy them at run time, while MobileFirst Server is still serving traffic.

Changes in the adapter descriptor file

The `adapter.xml` descriptor file for V8.0.0 has changed slightly.

- For more information on the structure of the adapter descriptor file for Java adapters, see "The Java adapter-descriptor file" on page 7-194.
- For more information on the structure of the adapter descriptor file for JavaScript adapters, see "The JavaScript adapter-descriptor file" on page 7-206.

Integration with Swagger UI

Starting from V8.0.0, MobileFirst Server integrates with Swagger UI. For any adapter, you can view the associated API by clicking **View Swagger**

Docs in the **Resources** tab in the MobileFirst Operations Console. The feature is available in development environments only.

Support for JavaScript adapters
V8.0.0 supports JavaScript adapters with HTTP and SQL connectivity types, only.

Support for JAX-RS 2.0
JAX-RS 2.0 introduces new server-side functionality: server-side asynchronous HTTP, filters and interceptors. MobileFirst adapters can now exploit these new features. For more information see "Implementing the JAX-RS service of the adapter" on page 7-200.

IBM MobileFirst Platform Foundation on IBM Containers

IBM MobileFirst Platform Foundation on IBM Containers released for V8.0.0 is available on the IBM Passport Advantage® site. This version of IBM MobileFirst Platform Foundation on IBM Containers is production ready and supports enterprise dashDB™ transactional database on IBM Bluemix.

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

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

Deploying MobileFirst Server V8.0.0 on IBM PureApplication System

You can now deploy and configure MobileFirst Server V8.0.0 to the supported IBM MobileFirst Platform Foundation System Pattern on IBM PureApplication System.

All supported IBM MobileFirst Platform Foundation System Pattern now include support for an existing IBM DB2® database.

IBM MobileFirst Platform Application Center is now supported on a Virtual System Pattern.

Learn more about Deploying MobileFirst Server on IBM PureApplication System.

What's new in MobileFirst Server

MobileFirst Server has been redesigned to help reduce the time and cost of deploying and updating your apps. In addition to the redesign of the MobileFirst Server, IBM MobileFirst Platform Foundation expands the number of installation methods available.

The new MobileFirst Server design introduces two new components, MobileFirst Server live update service and MobileFirst Server artifacts.

MobileFirst Server live update service is designed to help reduce the time and cost of incremental updates for your apps. It manages and stores the server-side configuration data of the apps and adapters. You can change or update various parts of your app with rebuilding or redeploying your app:

- Dynamically change or update app behavior based on user segments that you define.
- Dynamically change or update server-side business logic.
- Dynamically change or update app security

- Externalize and dynamically change app configuration.

MobileFirst Server artifacts provide resources for MobileFirst Operations Console. For more information about how the components function, see “MobileFirst Server overview” on page 6-2.

Along with the redesign of MobileFirst Server, more installation options are now provided. In addition to the manual installation, IBM MobileFirst Platform Foundation gives you two options to install MobileFirst Server in a server farm. You can also install MobileFirst Server in Liberty collective.

Starting with V8.0, you can now install the MobileFirst Server components in a server farm by using Ant tasks, or with the Server Configuration Tool. For more information, see the following topics:
- “Installing a server farm” on page 6-139
- “Tutorials about MobileFirst Server installation” on page 6-4

MobileFirst Server V8.0 also supports Liberty collective. For more information about the server topology and various installation methods, see the following topics:
- “Liberty collective topology” on page 6-91
- “Running the Server Configuration Tool” on page 6-106
- “Installing with Ant Tasks” on page 6-110
- “Manual installation on WebSphere Application Server Liberty collective” on page 6-121

What's new in IBM MobileFirst Analytics

IBM MobileFirst Analytics introduces a redesigned console with information presentation improvements and role-based access controls. The console is also now available in a number of different languages.

The MobileFirst Analytics Console was redesigned to present information in an intuitive and more meaningful fashion, and uses summarized data for some event types.

You can now sign out of the MobileFirst Analytics Console by clicking on the gear icon.

The MobileFirst Analytics Console is now available in the following languages:
- German
- Spanish
- French
- Italian
- Japanese
- Korean
- Portuguese (Brazilian)
- Russian
- Simplified Chinese
- Traditional Chinese

The MobileFirst Analytics Console now shows different content based on the security role of the logged-in user.

For more information, see “Role-based access control” on page 11-23.

Starting with V8.0.0, the MobileFirst Analytics Server uses Elasticsearch Version 1.7.5.

For V8.0.0 Analytics support for web applications was added with the new web analytics client-side API. See individual topics in the “Developing the analytics client” on page 11-35 section for details of available functionality.

Some event types were changed between earlier versions of MobileFirst Analytics Server and V8.0.0. Because of this change, any JNDI properties that were previously configured in your server configuration file must be converted to the new event type.

For more information, see “Migration of server properties used by previous versions of MobileFirst Analytics Server” on page 11-12.

What's new in push notifications

With IBM MobileFirst Platform FoundationV8.0.0, the push notification service is provided as a stand-alone service hosted on a separate web application.

Earlier versions of IBM MobileFirst Platform Foundation embedded the push notification service as part of the application runtime.

Programming model

The programming model spans across the server to client, and you need to set up your application for push notification service to work on your client applications. Two types of clients would interact with push notification service:

- Mobile client applications
- Back-end server applications

Security for push notification service

IBM MobileFirst Platform Foundation authorization server enforces the OAuth protocol to secure push notification service.

For more information, see “Security for push notification clients” on page 7-251.

Push notification service model

With IBM MobileFirst Platform Foundation V8.0.0, the event source-based model is not supported. The push notification capability is enabled on IBM MobileFirst Platform Foundation by the push service model.

Push REST API

You can enable back-end server applications that are deployed outside MobileFirst Server to access push notification functions by using REST API for push in the IBM MobileFirst Platform Foundation runtime. For information on using REST API for push notification, see “REST API for the MobileFirst Server push service” on page 8-197.

Upgrading from existing event source-based notification model

With the IBM MobileFirst Platform Foundation V8.0.0, the event source-based model is not supported. The push notification capability is enabled entirely by the push service model. All existing event source-based applications need to be migrated to the new push service model. For information on migrating your push notifications, see "Migrating to push notifications from event source-based notifications" on page 5-54.

Sending push notifications

You can choose to send an event-source based, tag-based, or broadcast-enabled push notification from the server.

Push notifications can be sent by using the following methods:
- Using MobileFirst Operations Console, two types of notifications can be sent: `tag` and `broadcast`. See "Sending push notification with the MobileFirst Operations Console" on page 7-259.
- Using "Push Message (POST)" on page 8-236 REST API, all forms of notifications can be sent: `tag`, `broadcast`, and `authenticated`.
- Using "REST API for the MobileFirst Server administration service" on page 8-7, all forms of notifications can be sent: `tag`, `broadcast`, and `authenticated`.

Sending SMS notifications

You can configure the push service to send a short message service (SMS) notification to user devices.

Refer Sending SMS notifications, for more information.

Installation of the push notification service

The push notification service is packaged as a MobileFirst Server component (MobileFirst Server push service). For more information about the architecture of MobileFirst Server, see "MobileFirst Server overview" on page 6-2.

To find out how to install the push service and other MobileFirst Server components, see "Installing MobileFirst Server for a production environment" on page 6-39.

Push service model is supported on Windows Universal Platform apps

You can now migrate native Windows Universal Platform (UWP) applications to use the push service model to send push notifications.

Refer "Native Windows Universal applications" on page 5-75, for more information.

What's new in V8.0.0 interim fixes

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

Interim fixes are cumulative. When you download the latest V8.0.0 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.

Licensing

PVU licensing

A new offering, IBM MobileFirst Platform Foundation Extension V8.0.0, is available through PVU (processor value unit) licensing. For more information on PVU licensing for IBM MobileFirst Platform Foundation Extension, see Licensing MobileFirst.

Web applications

Registering web applications from the MobileFirst Platform CLI (APAR PI65327)

You can now register client web applications to MobileFirst Server by using the IBM MobileFirst Platform Command Line Interface (CLI) (**`mfpdev`**) as an alternative to registration from the IBM MobileFirst Platform Operations Console. For more information, see "Registering web applications from the MobileFirst Platform CLI" on page 7-80.

Adapters

Added `mfpdev` push and pull commands for Java and JavaScript adapter configurations

You can use IBM MobileFirst Platform Command Line Interface (CLI) to push Java and JavaScript adapter configurations to the MobileFirst Server and pull adapter configurations from the MobileFirst Server. See "Pushing Java adapter configurations" on page 7-199, "Pulling Java adapter configurations" on page 7-199, "Pushing JavaScript adapter configurations" on page 7-217, and "Pulling JavaScript adapter configurations" on page 7-217 for more information.

Commands for building or deploying multiple adapters

You can build all of the adapters within a directory by using the **`mfpdev adapter build all`** command. For more information about this command, see "Building JavaScript adapters" on page 7-215 or "Building Java adapters" on page 7-197.

You can deploy all of the adapters within a directory by using the **`mfpdev adapter deploy all`** command. For more information about this command, see "Deploying JavaScript adapters" on page 7-216 or "Deploying Java adapters" on page 7-198.

Cordova applications

Opening the native IDE for a Cordova project from Eclipse with the Studio plug-in

With the Studio plug-in installed in your Eclipse IDE, you can open an existing Cordova project in Android Studio or Xcode from the Eclipse interface to build and run the project. For more information, see "Opening a Cordova project in a platform development environment" on page 7-120.

Deprecated features and API elements

New releases of IBM MobileFirst Platform Foundation might introduce features or API elements that supersede features and API elements from past releases. Superseded features and API elements are deprecated and they might be removed in future releases.

Review the following deprecated features and API elements to determine the impact that these deprecations might have on your your IBM MobileFirst Platform Foundation environment.

API elements deprecated in V8.0.0

- For more information about deprecated server-side API elements, see Java API elements deprecated in V8.0.0.
- For more information about deprecated client-side JavaScript API elements, see Deprecated JavaScript APIs.
- For more information about discontinued and deprecated Windows C# API elements, see Deprecated Windows C# API elements.

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.	
IBM MobileFirst Mobile Patterns	IBM MobileFirst Mobile Patterns are deprecated in MobileFirst Studio.	

Features deprecated in V7.0.0

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

Category	Deprecation	Recommended Action
BlackBerry 6 and BlackBerry 7 environments	BlackBerry 6 and BlackBerry 7 environments are deprecated.	

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

Category	Deprecation	Recommended Action
Analytics Reports database	The Reports database, often referenced as `WLREPORT` in the documentation, is deprecated in IBM MobileFirst Platform Foundation V7.0.0.	Use IBM MobileFirst Analytics instead. Note that setting up the Reports database is optional in this release and prior releases. Also note that the use of the Reports database is redundant with MobileFirst Analytics in this release and recent prior releases.
Analytics BIRT predefined reports	The predefined BIRT reports are deprecated.	Use IBM MobileFirst Analytics console and custom chart support instead.
The JAR files and JavaScript libraries that enable SSO between IBM MobileFirst Platform Foundation 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 "Overview of the MobileFirst security framework" on page 7-265.

API elements deprecated in V7.0.0

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

Category	Deprecation	Recommended Action
Objective-C: WLClient	[WLCLient lastAccessToken]	Use [WLAuthorizationManager cachedAuthorizationHeader] instead.
	[WLCLient lastAccessTokenForScope]	Use [WLAuthorizationManager cachedAuthorizationHeader] instead.
	[WLCLient obtainAccessTokenForScope]	Use [WLAuthorizationManager obtainAuthorizationHeaderForScope] instead.
Objective-C: WLResponse	[WLResponse getResponseJson]	Use the responseJson property instead.
Java	WLClient.obtainAccessToken (String scope, WLResponseListener responseListener)	Use the WLAuthorizationManager class instead.
	WLClient.getLastAccessToken	Use the WLAuthorizationManager class instead.
	WLClient.obtainAccessToken (String scope, WLResponseListener responseListener, WLRequestOptions requestOptions)	Use the WLAuthorizationManager class instead.
	WLClient.getRequiredAccessTokenScope (int status, String header)	Use the WLAuthorizationManager class instead.
	WLClient.logActivity	Use the Logger class instead.

Table 3-3. API elements deprecated in V7.0.0 (continued).

Category	Deprecation	Recommended Action
JavaScript	WLClient.obtainAccessToken	Use the WLAuthorizationManager class instead.
	WLClient.getRequiredAccessTokenScope	Use the WLAuthorizationManager class instead.
	WLClient.getLastAccessToken	Use the WLAuthorizationManager class instead.
	WLClient.logActivity	Use the WLLogger class 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.	

Discontinued features and API elements

Consider carefully how removed features and API elements affect your IBM MobileFirst Platform Foundation environment.

Features that are discontinued in V8.0.0 and features that are not included in V8.0.0

IBM MobileFirst Platform Foundation V8.0.0 is radically simplified compared to the previous version. As a result of this simplification, some features that were available in V7.1 are discontinued in V8.0.0. In most cases, an alternative way to implement the features is suggested. These features are marked *discontinued*. Some other features that exist in V7.1. are not in V8.0.0, but not as a consequence of the new design of V8.0.0. To distinguish these excluded features from the features that are discontinued from V8.0.0, they are marked *not in V8.0.0*.

Table 3-5. Features that are discontinued in V8.0.0 and features that are not included in V8.0.0

Feature	**Status and replacement path**
MobileFirst Studio is replaced by MobileFirst Studio plug-in for Eclipse.	Replaced by MobileFirst Studio plug-in for Eclipse empowered by standard and community-base Eclipse plug-ins. You can develop hybrid applications directly with the Apache Cordova CLI or with a Cordova enabled IDE such as Visual Studio Code, Eclipse, IntelliJ, and others.For more information about using eclipse as a Cordova enabled IDE, see "IBM MobileFirst Studio plug-in for managing Cordova projects in Eclipse" on page 7-114. You can develop adapters with Apache Maven or a maven-enabled IDE such as Eclipse, IntelliJ, and others. For more information about developing adapters, see "Adapters as Apache Maven projects" on page 7-189. For more information about using Eclipse as a Maven enabled IDE, read the Developing Adapters in Eclipse tutorial. Install IBM MobileFirst Platform Foundation Developer Kit to test adapters and applications with MobileFirst Development Server. You can also access MobileFirst development tools and SDKs if you do not want to download them from Internet-based repositories such as NPM, Maven, Cocoapod, or NuGet. For more information about IBM MobileFirst Platform Foundation Developer Kit, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.
Skins, Shells, the Setting page, minification, and JavaScript UI elements are discontinued for hybrid applications.	Discontinued. Hybrid applications are developed directly with the Apache Cordova. For more information about replacing skins, shells, the Setting page, and minification, see "Removed components" on page 5-49 and "Comparison of Cordova apps developed with V8.0.0 versus V7.1.0 and before" on page 5-34. For more information about replacing the JavaScript UI elements, see Discontinued JavaScript UI elements.
Sencha Touch can no longer be imported into MobileFirst projects for hybrid applications.	Discontinued. MobileFirst hybrid applications are developed directly with the Apache Cordova, and the MobileFirst features are provided as Cordova plug-ins. Refer to the Sencha Touch documentation to integrate Sencha Touch and Cordova. For more information about using IBM MobileFirst Platform Foundation with Cordova, see "Developing Cordova applications" on page 7-83.

Table 3-5. Features that are discontinued in V8.0.0 and features that are not included in V8.0.0 (continued)

Feature	Status and replacement path
The encrypted cache is discontinued.	Discontinued To store encrypted data locally, use JSONStore. For more information about JSONStore, see "JSONStore overview" on page 7-134.
A sample that shows how to set up Touch ID support for JSONStore is not in V8.0.0.	Not in V8.0.0. **Note:** You can use the example that is provided in the documentation of version 7.1 and adapt it to the code of your client application. For more information about the example in version 7.1, see Setting up Touch ID support for JSONStore (version 7.1).
Triggering Direct Update on demand is not in V8.0.0. The client application checks for Direct Update when it obtains the OAuth token for a session. You cannot program a client application to check for direct updates at a different point in time in V8.0.0.	Not in V8.0.0. **Note:** For more information about customizing Direct Update, see "Customizing the Direct Update user interface and process" on page 7-244.
Adapters with session-dependency configuration. In V7.1.0, you can configure MobileFirst Server to work in session-independent mode (default) or in session-dependent mode. Beginning with V8.0.0, session-dependent mode is no longer supported. The server is inherently independent of the HTTP session, and no related configuration is required.	Discontinued.
Attribute store over IBM WebSphere eXtreme Scale is not supported in V8.0.0.	Not in V8.0.0.
Service discovery and adapter generation for IBM® Business Process Manager (IBM BPM) process applications, Microsoft Azure Marketplace DataMarket, OData RESTful APIs, RESTful resources, Services that are exposed by an SAP Netweaver Gateway, and Web Services is not in V8.0.0.	Not in V8.0.0.
The JMS JavaScript adapter is not in V8.0.0.	Not in V8.0.0.
The SAP Gateway JavaScript adapter is not in V8.0.0.	Not in V8.0.0.
The SAP JCo JavaScript adapter is not in V8.0.0.	Not in V8.0.0.
The Cast Iron® JavaScript adapter in not in V8.0.0.	Not in V8.0.0.
The OData and Microsoft Azure OData JavaScript adapters are not in V8.0.0.	Not in V8.0.0.

Table 3-5. Features that are discontinued in V8.0.0 and features that are not included in V8.0.0 (continued)

Feature	**Status and replacement path**
Push notification support for USSD is not supported in V8.0.0.	Discontinued
JMS-based push notification is not supported in V8.0.0.	Discontinued
Event-based push notifications is not supported in V8.0.0.	Discontinued. Use the push notification service. For more information on migrating to push notification service, see topic "Migrating to push notifications from event source-based notifications" on page 5-54.
Security: User-certificate authentication. V8.0.0 does not include any predefined security check to authenticate users with X.509 client-side certificates.	Not in V8.0.0.
Security: Simple data sharing. The WLSimpleDataSharing API is not included in V8.0.0. Therefore, using simple data sharing to configure device single sign-on with a reverse proxy (for example to exchange LTPA tokens between applications) is also not supported in V8.0.0.	Not in V8.0.0.
Security: Integration with IBM Trusteer®. V8.0.0 does not include any predefined security check or challenge to test IBM Trusteer risk factors.	Not in V8.0.0. Use IBM Trusteer Mobile SDK.
Security: Device provisioning and device auto-provisioning.	Discontinued. **Note:** Device provisioning is handled in the normal authorization flow. Device data is automatically collected during the registration process of the security flow. For more information about the security flow, see "End-to-end authorization flow" on page 7-266
Security: Configuration file for obfuscating Android code with ProGuard. V8.0.0 does not include the predefined `proguard-project.txt` configuration file for Android ProGuard obfuscation with a MobileFirst Android application.	Not in V8.0.0.
Security: Adapter based authentication is replaced. Authentication uses the OAuth protocol and is implemented with security checks.	Replaced by a security check based implementation. **Note:** For an introduction to the authentication and security in V8.0.0, see Authorization Concepts.

Table 3-5. Features that are discontinued in V8.0.0 and features that are not included in V8.0.0 (continued)

Feature	Status and replacement path
Security: LDAP login. V8.0.0 does not include any predefined security check to authenticate users with an LDAP server. Instead, for WebSphere Application Server or WebSphere Application Server Liberty use the application server or a gateway to map an Identity Provider such as LDAP to LTPA, and generate an OAuth token for the user by using an LTPA security check.	Not in V8.0.0. Replaced by an LTPA security check for WebSphere Application Server or WebSphere Application Server Liberty. **Note:** For an example of security-check that uses LDAP and LTPA with MobileFirst Server running on WebSphere Application Server or WebSphere Application Server Liberty, read the Working with LDAP and LTPA in IBM MobileFirst Platform Foundation 8.0 blog. **Note:** For more information about the predefined LTPA security check, see "LTPA-based single sign-on (SSO) security check" on page 7-285.
Authentication configuration of the HTTP adapter. The predefined HTTP adapter does not support the connection as a user to a remote server. **Note:** For more information about the HTTP adapter configuration, see "HTTP adapter connectionPolicy element" on page 7-209.	Not in V8.0.0. Edit the source code of the HTTP adapter and add the authentication code. Use MFP.Server.invokeHttp to add identification tokens to the HTTP request's header.
Security Analytics, the ability to monitor MobileFirst security framework's events with MobileFirst Analytics Console is not in V8.0.0.	Not in V8.0.0.
The event source-based model for push notifications is discontinued and replaced by the tag-based push service model.	Discontinued and replaced by the tag-based push service model. For more information about migrating event source-based notifications to the push service model, see "Migrating to push notifications from event source-based notifications" on page 5-54.
Unstructured Supplementary Service Data (USSD) support is not in V8.0.0.	Not in V8.0.0.
Cloudant® used as a database for MobileFirst Server in not supported in V8.0.0.	Not in V8.0.0.
Geolocation: The geolocation support is discontinued in IBM MobileFirst Platform Foundation V8.0.0. The REST API for beacons and for mediators is discontinued. The client-side and server-side API WL.Geo and WL.Device are discontinued.	Discontinued. Use the native device API or third-party Cordova plug-ins for geolocation.
The MobileFirst Data Proxy feature is discontinued. The Cloudant IMFData and CloudantToolkit APIs are also discontinued.	Discontinued. For more information about replacing the IMFData and CloudantToolkit APIs in your apps, see "Migrating apps storing mobile data in Cloudant with IMFData or Cloudant SDK" on page 5-80.

Table 3-5. Features that are discontinued in V8.0.0 and features that are not included in V8.0.0 (continued)

Feature	Status and replacement path
The IBM Tealeaf® SDK is no longer bundled with IBM MobileFirst Platform Foundation.	Discontinued. Use IBM Tealeaf SDK. For more information, see Tealeaf installation and implementation in an Android application and Tealeaf iOS Logging Framework Installation and Implementation in the IBM Tealeaf Customer Experience documentation.
IBM MobileFirst Platform Test Workbench is not bundled with IBM MobileFirst Platform Foundation	Discontinued.
BlackBerry, Adobe AIR, Windows Silverlight are not supported by IBM MobileFirst Platform Foundation V8.0.0. No SDK is provided for these platforms.	Discontinued.

Discontinued API elements in V8.0.0

- For more information about discontinued client-side API, see "Client API changes in V8.0.0" on page 5-4.
- For more information about discontinued server-side API, see "Server-side API changes in V8.0.0" on page 5-14.

Features discontinued in V7.1.0

No features were removed in V7.1.0.

Features discontinued in V7.0.0

Table 3-6. Features discontinued in V7.0.0

Feature
The JAR files and JavaScript libraries that enable SSO between IBM MobileFirst Platform Foundation and other external services are removed. Use the MobileFirst OAuth-based security model instead. For more information about the OAuth-based security model, see "Overview of the MobileFirst security framework" on page 7-265.
The "shake to refresh" feature is removed in IBM MobileFirst Platform Foundation V7.0.0.

Features discontinued in V6.3.0

Table 3-7. Features discontinued in V6.3.0

Feature
The IBM Worklight® Application Framework (beta) set of tools is removed in IBM MobileFirst Platform Foundation 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=SSVNUQ&tc=SSHT2F&atrn=SWVersion&atrv=8.0

Known limitations

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

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

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

Note: For more information about product known limitations or issues, see "Known issues."

Globalization

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

- Partial translation: Part of the product IBM MobileFirst Platform Foundation V8.0.0, including its documentation, is translated in the following languages: Simplified Chinese, Traditional Chinese, French, German, Italian, Japanese, Korean, Portuguese (Brazil), Russian, and Spanish. User-facing text is translated.
- Bidirectional support: The applications that are generated by IBM MobileFirst Platform Foundation 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, no hard dependency exists 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 core functionality, some GUI elements are not mirrored.
- Constraints on adapter names: Names of adapters must be valid names to create a Java class name. In addition, they must be composed only of the following characters:
  - Uppercase and lowercase letters (A-Z and a-z)
  - Digits (0-9)
  - Underscore (_)
- Unicode characters: Unicode characters outside the Basic Multilingual Plane are not supported.
- Language sensitivity and Unicode Normalization Forms: In the following use cases, queries do not consider language sensitivity such as normal matching,

accent-insensitive, case-insensitive, and 1-to-2 mapping for the search function to run correctly in different languages, and search on data does not use Normalization Form C (NFC).

– From the MobileFirst Analytics Console, when you create a custom filter for a custom chart. However, in this console, the message property uses Normalization Form C (NFC) and considers language sensitivity.
– From the MobileFirst Operations Console, when you search for an application in the Browse Applications page, for an adapter in the Browser Adapters page, for a tag in the Push page, or a device on the Devices page.
– In the Find functions for the JSONStore API.

IBM MobileFirst Analytics

The IBM MobileFirst Analytics has the following limitations:

- Security analytics (data on requests failing security checks) is not supported.
- In MobileFirst Analytics Console, the format for numbers does not follow the International Components for Unicode (ICU) rules.
- In MobileFirst Analytics Console, the numbers do not use the user's preferred number script.
- In MobileFirst 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 an alert in the Alert Management page of MobileFirst Analytics Console, the numerical data must be in base 10, Western, or European numerals, such as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- 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
- The Analytics client SDK is not available for Windows.

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 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 left-to-right language, the whole text direction is automatically left-to-right.

  The sequence of characters and text alignment do not match cultural fashion in the bidirectional language.

- The numeric fields do not parse numeric values according to the formatting rules of the locale. The console displays formatted numbers but accept as input only *raw* (unformatted) numbers. For example: `1000`, not `1 000`, or `1,000`.

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 WebSphere Application Server (full or Liberty profiles) does not support non-ASCII passwords and user names. See Characters that are valid for user IDs and passwords.
- On Windows:
  - To see any localized messages in the log file that the test server creates, you must open this log file with the UTF8 encoding.

  These limitations exist because of the following causes:
  - The test server is installed on WebSphere Application Server Liberty profile, which creates log file with the ANSI encoding except for its localized messages for which it uses the UTF8 encoding.
- 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.
- 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.

Adapters and third-party dependencies

The following known issues pertain to interactions between dependencies and classes in the application server, including the MobileFirst shared library.

Apache HttpClient
IBM MobileFirst Platform Foundation uses Apache HttpClient internally. If you add an Apache HttpClient instance as a dependency to a Java adapter, the following APIs do not work properly in the adapter: AdaptersAPI.executeAdapterRequest, AdaptersAPI.getResponseAsJSON, and AdaptersAPI.createJavascriptAdapterRequest. The reason is that the APIs contain Apache HttpClient types in their signature. The workaround is to use the internal Apache HttpClient but to change the dependency scope in the `pom.xml` to `provided`.

Bouncy Castle cryptographic library
IBM MobileFirst Platform Foundation uses Bouncy Castle itself. It might be possible to use another version of Bouncy Castle in the adapter, but the consequences need to be carefully tested: sometimes, the MobileFirst Bouncy Castle code populates certain static Singleton fields of the javax.security package classes and might prevent the version of Bouncy Castle that is inside an adapter from using features that rely on those fields.

Apache CXF implementation of JAR files
CXF is used in the MobileFirst JAX-RS implementation, thus preventing you from adding Apache CXF JAR files to an adapter.

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

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

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

Application Center requires MobileFirst Studio V7.1 for importing and building the Application Center mobile client

To build the Application Center mobile client, you need MobileFirst Studio V 7.1. You can download MobileFirst Studio from the Downloads page of the Developer Center website. Click the **Previous MobileFirst Platform Foundation releases** tab for the download link. For installation instructions, see Installing MobileFirst Studio in the IBM Knowledge Center for 7.1. For more information about building the Application Center mobile client, see “Preparations for using the mobile client” on page 13-6.

Application Center and Microsoft Windows Phone 8.1

Application Center supports the distribution of applications as Windows Phone application package (`.xap`) files for Microsoft Windows Phone 8.0 and Microsoft Windows Phone 8.1. With Microsoft Windows Phone 8.1, Microsoft introduced a new universal format as app package (`.appx`) files for Windows Phone. Currently, Application Center does not support the distribution of app package (`.appx`) files for Microsoft Windows Phone 8.1, but is limited to Windows Phone application package (`.xap`) files only.

Application Center supports the distribution of app package (`.appx`) files for Microsoft Windows Store (Desktop applications) only.

Administering MobileFirst applications through Ant or through the command line

The `mfpadm` tool is not available if you download and install only the IBM MobileFirst Platform Foundation Developer Kit. The `mfpadm` tool is installed with the MobileFirst Server with the installer.

Direct Update

Direct Update on Windows is not supported in V8.0.0.

FIPS 140-2 feature limitations

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

- This FIPS 140-2 validated mode applies only to the protection (encryption) of local data that is stored by the JSONStore feature and protection of HTTPS communications between the 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.
- This feature is only supported on the iOS and Android platforms.
  - On Android, this feature is only supported on devices or simulators that use the **`x86`** or **`armeabi`** architectures. It is not supported on Android using **`armv5`** or **`armv6`** architectures. The reason is because the OpenSSL library used did not obtain FIPS 140-2 validation for **`armv5`** or **`armv6`** on Android. FIPS 140-2 is not supported on 64-bit architecture even though the MobileFirst library does support 64-bit architecture. FIPS 140-2 can be run on 64-bit devices if the project includes only 32-bit native NDK libraries.
  - On iOS, it is supported on **`i386`**, **`x86_64`**, **`armv7`**, **`armv7s`**, and **`arm64`** architectures.
- This feature works with hybrid applications only (not with native applications).
- For native iOS, FIPS is enabled through the iOS FIPS libraries and is enabled by default. No action is required to enable FIPS 140-2.
- 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 10-75.

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.

JSONStore supported architectures

For Android, JSONStore supports the following architectures: ARM, ARM v7, and x86 32-bit. Other architectures are not currently supported. Trying to use other architectures leads to exceptions and potential application crashes.

JSON Store is not supported for Windows native applications.

For more information about JSONStore, see “JSONStore overview” on page 7-134.

Liberty server limitations

If you use the Liberty 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 computer, you might configure JVM preferences to `mx512m` and `Xms216m`.

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 authenticated as anonymous has attempted to access a session owned by {user name}
```

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.

Microsoft Windows Phone 8

For Windows Phone 8.1 environments, x64 architecture is not supported.

Microsoft Windows 10 UWP apps

Application authenticity feature does not work on MobileFirst Windows 10 UWP apps when the MobileFirst SDK is installed through the NuGet package. As a workaround, developers can download the NuGet package and add the MobileFirst SDK references manually.

MobileFirst Operations Console

Analytics page

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 Analytics clustering. Consider testing your load before you integrate IBM MobileFirst Analytics into production.

Nested projects can result in unpredictable results with the CLI

Do not nest projects inside one another when using the IBM MobileFirst Platform Command Line Interface (CLI). Otherwise, the project that is acted upon might not be the one that you expect.

Previewing Cordova web resources with the Mobile Browser Simulator

You can preview your web resources with Mobile Browser Simulator, but not all MobileFirst JavaScript APIs are supported by the simulator. In particular, the OAuth protocol is not fully supported. However, you can test calls to adapters with WLResourceRequest. For more information, see “Previewing Cordova web resources with the Mobile Browser Simulator” on page 7-109.

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.

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.

Support for push notification

Non-secured push is supported in Cordova (on iOS and Android).

Updating the cordova-ios platform

To update the cordova-ios platform of a Cordova app, you must uninstall and reinstall the platform by completing the following steps:

1. Navigate to the project directory for the app by using the command-line interface.
2. Run the `cordova platform rm ios` command to remove the platform.
3. Run the `cordova platform add ios@`*`version`* command to add the new platform to the app, where *version* is the version of the Cordova iOS platform.
4. Run the `cordova prepare` command to integrate the changes.

The update fails if you use the `cordova platform update ios` command.

WKWebView support for iOS Cordova applications

App notifications and Direct Update features might not work well in iOS Cordova apps with WKWebView.

This limitation is due to the defect file:// url XmlHttpRequests are not allowed in WKWebViewEgine in `cordova-plugin-wkwebview-engine`.

To circumvent this issue, run the following command in your Cordova project.

```
cordova plugin add https://github.com/apache/cordova-plugins.git#master:wkwebview-engine-localhost
```

Executing this command would run a local web server in your Cordova application, you can then host and access your local files instead of using the file URI scheme (`file://`) to work with local files.

Note: This Cordova plug-in is not published to the Node package manager (`npm`).

`cordova-plugin-statusbar` does not work with Cordova application loaded with `cordova-plugin-mfp`.

`cordova-plugin-statusbar` will not work with Cordova application loaded with `cordova-plugin-mfp`.

To circumvent this issue, the developer will have to set CDVViewController as the root view controller. Replacing the code snippet in the wlInitDidCompleteSuccessfully method as suggested below in the `MFPAppdelegate.m` file of the Cordova iOS project.

Existing code snippet:

```
(void)wlInitDidCompleteSuccessfully
{
UIViewController* rootViewController = self.window.rootViewController;
// Create a Cordova View Controller
CDVViewController* cordovaViewController = [[CDVViewController alloc] init] ;
cordovaViewController.startPage = [[WL sharedInstance] mainHtmlFilePath];
// Adjust the Cordova view controller view frame to match its parent view bounds
cordovaViewController.view.frame = rootViewController.view.bounds;
// Display the Cordova view [rootViewController addChildViewController:cordovaViewController];
[rootViewController.view addSubview:cordovaViewController.view];
[cordovaViewController didMoveToParentViewController:rootViewController];
}
```

Recommended code snippet with workaround for the limitation:

```
(void)wlInitDidCompleteSuccessfully
{
 // Create a Cordova View Controller
CDVViewController* cordovaViewController = [[CDVViewController alloc] init] ;
cordovaViewController.startPage = [[WL sharedInstance] mainHtmlFilePath];
[self.window setRootViewController:cordovaViewController];
[self.window makeKeyAndVisible];
}
```

Raw IPV6 address not supported in Android applications

During the configuration of `mfpclient.properties` for your native Android application, if your MobileFirst Server is on a host with IPV6 address, then use a mapped host name for the IPV6 address to configure the **`wlServerHost`** property in `mfpclient.properties`. Configuring the **`wlServerHost`** with raw IPV6 address fails the application's attempt to connect to the MobileFirst Server.

Tutorials and additional resources

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

Tutorials

For you to get started with the most important features of IBM MobileFirst Platform Foundation, tutorials are available on the All Tutorials page of the Developer Center website.

Additional documentation

The Learn More page of the Developer Center website for IBM MobileFirst Platform provides useful links, including the Scalability and Hardware Sizing document and its accompanying hardware calculator spreadsheet.

You can find further helpful community resources here:

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

Upgrading to IBM MobileFirst Platform Foundation V8.0.0

This section contains the instructions for migrating the applications that you created with IBM MobileFirst Platform Foundation V6.2 or later to IBM MobileFirst Platform Foundation V8.0.0.

Migrating apps from earlier releases

IBM MobileFirst Platform Foundation V8.0 introduces new concepts for application development and deployment, and some API changes. Learn about these changes to prepare and plan for the migration of your MobileFirst applications.

At a glance

1. Learn about the development and deployment process in V8.0.0. See "Changes in the development and deployment process."
2. Learn about the migration assistance tool. Depending on your platform, see "Starting the Cordova app migration with the migration assistance tool" on page 5-38, "Scanning existing MobileFirst native iOS apps to prepare for MobileFirst version 8.0" on page 5-17, "Scanning existing MobileFirst native Android apps to prepare for MobileFirst version 8.0" on page 5-25, or "Scanning existing MobileFirst native Windows apps to prepare for MobileFirst version 8.0" on page 5-30.
3. Migrate your application. See "Migrating a Cordova or hybrid application" on page 5-2 or "Migrating a native application" on page 5-2.
4. Migrate adapters and security. See "Migrating adapters and security" on page 5-3.
5. Migrate push notifications. See "Migrating push notification support" on page 5-3.

Changes in the development and deployment process

For a quick hands-on experience of the development process with IBM MobileFirst Platform Foundation V8.0.0, you can review the Quick Start tutorials.

In this version of the product, you no longer create a project WAR file that needs to be installed in the application server that is running MobileFirst Server before you can upload your apps. Instead, MobileFirst Server is installed once, and you upload the server-side configuration of your apps, of the resource security, or of the push service to the server. You can modify the configuration of your apps with the MobileFirst Operations Console. You can also upload a new configuration file for your apps by using a command-line tool or the server REST API.

MobileFirst projects no longer exist. Instead, you develop your mobile app with the development environment of your choice. You develop the server-side of your application separately, in Java or in JavaScript. For more information about development environments for the client-side of your application, see "Client app development environments" on page 7-8. You can develop adapters with Apache Maven or a Maven enabled IDE such as Eclipse, IntelliJ, and others. For more information about developing adapters, see "Adapters as Apache Maven projects" on page 7-189. For more information about using Eclipse as a Maven enabled IDE, read the Developing Adapters in Eclipse tutorial.

In previous versions, applications were deployed to the server by uploading a `.wlapp` file. The file contained data that described the application and for hybrid applications, the web resources. In V8.0.0, the `.wlapp` file is replaced by an application descriptor JSON file for registering an app to the server. For Cordova applications that use Direct Update, instead of uploading a new version of the `.wlapp`, you now upload a web resource archive to the server.

When you develop your app, you use the IBM MobileFirst Platform Command Line Interface (CLI) for many tasks, such as registering an app to its target server or uploading its server-side configuration. For more information, see "Getting started with the MobileFirst CLI" on page 7-22.

Discontinued features and replacement path

IBM MobileFirst Platform Foundation V8.0.0 is radically simplified compared to the previous version. As a result of this simplification, some features that were available in V7.1 are discontinued in V8.0.0. For more information about discontinued features and replacement path, see "Features that are discontinued in V8.0.0 and features that are not included in V8.0.0" on page 3-19.

Migrating a Cordova or hybrid application

You start developing Cordova apps with the Apache Cordova command-line tool or with a Cordova enabled IDE such as Visual Studio Code, Eclipse, IntelliJ, and others.For more information about using eclipse as a Cordova enabled IDE, see "IBM MobileFirst Studio plug-in for managing Cordova projects in Eclipse" on page 7-114.

Add support for the MobileFirst features by adding the MobileFirst plug-ins to your app. For more information about the differences between V7.1 Cordova or hybrid apps and V8.0 Cordova apps, see "Comparison of Cordova apps developed with V8.0.0 versus V7.1.0 and before" on page 5-34.

To migrate a Cordova or hybrid app, you need to

- For planning purpose, run the migration assistance tool on your existing project. Review the generated report and assess the effort required for migration. For more information, see "Starting the Cordova app migration with the migration assistance tool" on page 5-38.
- Replace the client-side APIs that are discontinued or not in V8.0.0. For a list of API changes, see "Upgrading the WebView" on page 5-45.
- Modify the call to client resources that use the classic security model. For example, use the WLResourceRequest API, instead of WLClient.invokeProcedure, which is deprecated.
- If you use Direct Update, review "Migrating Direct Update" on page 5-44.

For more information about migrating Cordova or hybrid apps, see "Migrating existing Cordova and hybrid applications" on page 5-33.

Note: The migration of push notification support requires client-side and server-side changes and is described later on in "Migrating push notification support" on page 5-3.

Migrating a native application

To migrate native application, you need to follow these steps:

- For planning purpose, run the migration assistance tool on your existing project. Review the generated report and assess the effort required for migration.
- Update your project to use the SDK from IBM MobileFirst Platform Foundation V8.0.0
- Replace the client-side APIs that are discontinued or not in V8.0.0. The migration assistance tool can scan your code and generate reports of the APIs to replace.
- Modify the call to client resources that use the classic security model. For example, use the WLResourceRequest API, instead of WLClient.invokeProcedure, which is deprecated.

For more information about migrating native iOS apps, see “Migrating existing native iOS applications” on page 5-17.

For more information about migrating native Android apps, see “Migrating existing native Android applications” on page 5-24.

For more information about migrating native Windows apps, see “Migrating existing native Windows applications” on page 5-29.

Note: The migration of push notification support requires client-side and server-side changes and is described later on in “Migrating push notification support.”

Migrating adapters and security

Starting with V8.0.0, adapters are Maven projects. The MobileFirst security framework is based on OAuth, security scopes, and security checks. Security scopes define the security requirements to access a resource. Security checks define how a security requirement is verified. Security checks are written as Java or JavaScript adapters. For a hands-on experience with adapters and security, see the tutorials Creating Java and JavaScript Adapters and Authorization concepts.

MobileFirst Server operates only in session-independent mode and adapters should not store a state locally to a Java virtual machine (JVM).

You can externalize adapter properties to configure adapters for the context where they run, for example a test server or a production server. But the values of these properties are no longer included in a property file of a project WAR file. Instead, you define them from the MobileFirst Operations Console, or by using a command-line tool or the server REST API.

For more information about migrating adapters, see “Migrating existing adapters to work under MobileFirst Server V8.0.0” on page 5-51.

For more information about server-side API changes, see “Server-side API changes in V8.0.0” on page 5-14.

For an introduction to Apache Maven used to develop adapters, see “Adapters as Apache Maven projects” on page 7-189.

Migrating push notification support

The event-source-based model is no longer supported. Instead, use tag-based notification. To learn more about migrating push notification for your client apps and your server-side components, see "Migrating to push notifications from event source-based notifications" on page 5-54 and "Migration scenarios" on page 5-56.

Starting with V8.0.0, you configure the push service on the server side. The push certificates are stored on the server. You can set them from the MobileFirst Operations Console or you can automate certificate uploads by using a command-line tool or the push service REST API. You can also send push notifications from the MobileFirst Operations Console.

The push service is protected by the OAuth security model. You must configure server-side components that use the push service REST API must be configured as confidential clients of MobileFirst Server.

Changes in the server databases and in the server structure

MobileFirst Server enables changes to app security, connectivity and push without code change, app rebuild or redeployment. But these changes imply changes in the database schemas, the data stored in the database, and the installation process.

Because of these changes, IBM MobileFirst Platform Foundation does not include automated scripts to migrate your databases from earlier versions to V8.0.0 or to upgrade an existing server installation. To move new versions of your apps to V8.0.0, install a new server that you can run side by side with your previous server. Then, upgrade your apps and adapters to V8.0.0 and deploy them to the new server.

Storing mobile data in Cloudant

Storing mobile data in Cloudant with the IMFData framework or CloudantToolkit is no longer supported. For an alternative API, see "Migrating apps storing mobile data in Cloudant with IMFData or Cloudant SDK" on page 5-80.

Client API changes in V8.0.0

The following changes in the APIs are relevant to migrating your MobileFirst client application.

The following tables list the discontinued client-side API elements in V8.0.0, deprecated client-side API elements in V8.0.0, and suggested migration paths. For more information about migrating client applications, see "Migrating client applications to IBM MobileFirst Platform Foundation V8.0.0" on page 5-17.

JavaScript APIs

These JavaScript APIs that affect the user interface are no longer supported in v8.0. They can be replaced with available third-party Cordova plug-ins, or by creating custom Cordova plug-ins.

Table 5-1. Discontinued JavaScript UI elements

API element	Migration path

`WL.BusyIndicator`
`WL.OptionsMenu`
`WL.TabBar`
`WL.TabBarItem`
	Use Cordova plug-ins or HTML 5 elements.

`WL.App.close()`	Handle this event outside of MobileFirst.
`WL.App.copyToClipboard()`	Use Cordova plug-ins providing this functionality.
`WL.App.openUrl(url, target, options)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova `InAppBrowser` plug-in provides this feature..

`WL.App.overrideBackButton(callback)`
`WL.App.resetBackButton()`
	Use Cordova plug-ins providing this functionality.
	Note: For your information, the Cordova `backbutton` plug-in provides this feature..

`WL.App.getDeviceLanguage()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova cordova-plugin-globalization plug-in provides this feature.
`WL.App.getDeviceLocale()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova cordova-plugin-globalization plug-in provides this feature.
`WL.App.BackgroundHandler`	To run a custom handler function, use the standard Cordova pause event listener. Use a Cordova plug-in that provides privacy and prevents iOS and Android systems and users from taking snapshots or screen captures. For more information, see the description of the PrivacyScreenPlugin at https://github.com/devgeeks/PrivacyScreenPlugin.

`WL.Client.close()`
`WL.Client.restore()`
`WL.Client.minimize()`
	The functions were provided to support the Adobe AIR platform, which is not supported by IBM MobileFirst Platform V8.0.0.

`WL.Toast.show(string)`	Use Cordova plug-ins for Toast.

This set of APIs is no longer supported in V8.0.0.

Table 5-2. Other Discontinued JavaScript elements

API	Migration path
`WL.Client.checkForDirectUpdate(options)`	No replacement. **Note:** You can call WLAuthorizationManager.obtainAccessToken to trigger a direct update if one is available. The access to a security token triggers a direct update if one is available on the server. But you cannot trigger Direct Update on demand. For more information about customizing the Direct Update user interface and process, see "Customizing the Direct Update user interface and process" on page 7-244.

`WL.Client.setSharedToken({key: myName, value: myValue})`
 	
`WL.Client.getSharedToken({key: myName})`
 	
`WL.Client.clearSharedToken({key: myName})`
	No replacement.

`WL.Client.isConnected()`
 	
`connectOnStartup init option`
	Use WLAuthorizationManager.obtainAccessToken to check connectivity to the server and apply application management rules.

`WL.Client.setUserPref(key,value, options)`
 	
`WL.Client.setUserPrefs(userPrefsHash, options)`
 	
`WL.Client.deleteUserPrefs(key, options)`
	No replacement. You can use an adapter and the MFP.Server.getAuthenticatedUser API to manage user preferences.

`WL.Client.getUserInfo(realm, key)`
 	
`WL.Client.updateUserInfo(options)`
	No replacement.

`WL.Client.logActivity(activityType)`	Use WL.Logger. For more information, see "Logger SDK" on page 11-37.
`WL.Client.login(realm, options)`	Use WLAuthorizationManager.login. To get started with authentication and security, see the Authentication and Security tutorials.
`WL.Client.logout(realm, options)`	Use WLAuthorizationManager.logout.
`WL.Client.obtainAccessToken(scope, onSuccess, onFailure)`	Use WLAuthorizationManager.obtainAccessToken.

`WL.Client.transmitEvent(event, immediate)`
 	
`Wl.Client.purgeEventTransmissionBuffer()`
 	
`Wl.Client.setEventTransmissionPolicy(policy)`
	Create a custom adapter for receiving notifications of these events.

`WL.Device.getContext()`
 	
`WL.Device.startAcquisition(policy, triggers, onFailure)`
 	
`WL.Device.stopAcquisition()`
 	
`WL.Device.Wifi`
 	
`WL.Device.Geo.Profiles`
 	
`WL.Geo`
	Use native API or third-party Cordova plug-ins for GeoLocation.

`WL.Client.makeRequest (url, options)`	Create a custom adapter that provides the same functionality

Table 5-2. Other Discontinued JavaScript elements (continued)

API	Migration path
`WLDevice.getID(options)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, device.uuid from the cordova-plugin-device plug-in provides this feature.
`WL.Device.getFriendlyName()`	Use WL.Client.getDeviceDisplayName
`WL.Device.setFriendlyName()`	Use WL.Client.setDeviceDisplayName
`WL.Device.getNetworkInfo(callback)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the cordova-plugin-network-information plug-in provides this feature.
`WLUtils.wlCheckReachability()`	Create a custom adapter to check server availability.
`WL.EncryptedCache`	Use JSONStore to store encrypted data locally. JSONStore is in the cordova-plugin-mfp-jsonstore plug-in. For more information, see "JSONStore" on page 7-134.
`WL.SecurityUtils.remoteRandomString(bytes)`	Create a custom adapter that provides the same functionality.
`WL.Client.getAppProperty(property)`	You can retrieve the app version property by using the `cordova plugin add cordova-plugin-appversion` plug-in. The version that is returned is the native app version (Android and iOS only).
`WL.Client.Push.*`	Use "JavaScript client-side push API" on page 8-4 from the cordova-plugin-mfp-push plug-in. For more information, see "Migrating to push notifications from event source-based notifications" on page 5-54.
`WL.Client.Push.subscribeSMS(alias, adapterName, eventSource, phoneNumber, options)`	Use MFPPush.registerDevice(org.json.JSONObject options, MFPPushResponseListener listener) to register the device for push and SMS.
`WLAuthorizationManager.obtainAuthorizationHeader(scope)`	Use WLAuthorizationManager.obtainAccessToken to obtain a token for the required scope. For more information about implementing a custom resource request, see "JavaScript custom resource-request implementation sample" on page 7-311.
`WLClient.getLastAccessToken(scope)`	Use WLAuthorizationManager.obtainAccessToken

`WLClient.getLoginName()`
 	
`WL.Client.getUserName(realm)`
	No replacement

`WL.Client.getRequiredAccessTokenScope(status, header)`	Use WLAuthorizationManager.isAuthorizationRequired and WLAuthorizationManager.getResourceScope.
`WL.Client.isUserAuthenticated(realm)`	No replacement
`WLUserAuth.deleteCertificate(provisioningEntity)`	No replacement
`WL.Trusteer.getRiskAssessment(onSuccess, onFailure)`	No replacement

Table 5-2. Other Discontinued JavaScript elements (continued)

API	Migration path
WL.Client.createChallengeHandler(realmName)	To create a challenge handler for handling custom gateway challenges, use WL.Client.createGatewayChallengeHandler(gatewayName). To create a challenge handler for handling MobileFirst security-check challenges, use WL.Client.createSecurityCheckChallengeHandler(securityCheckName). For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
WL.Client.createWLChallengeHandler(realmName)	Use WL.Client.createSecurityCheckChallengeHandler(securityCheckName). For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
challengeHandler.isCustomResponse() where *challengeHandler* is a challenge-handler object that is returned by WL.Client.createChallengeHandler()	Use gatewayChallengeHandler.canHandleResponse() where *gatewayChallengeHandler* is a challenge-handler object that is returned by WL.Client.createGatewayChallengeHandler().
wlChallengeHandler.processSucccess() where *wlChallengeHandler* is a challenge-handler object that is returned by WL.Client.createWLChallengeHandler()	Use securityCheckChallengeHandler.handleSuccess() where *securityCheckChallengeHandler* is a challenge-handler object that is returned by WL.Client.createSecurityCheckChallengeHandler().
WL.Client.AbstractChallengeHandler.submitAdapterAuthentication()	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use a challenge-handler object that is returned by WL.Client.createGatewayChallengeHandler. For MobileFirst security-check challenge handlers, use a challenge-handler object that is returned by WL.Client.createSecurityCheckChallengeHandler.
WL.Client.AbstractChallengeHandler.submitFailure(err)	Use WL.Client.AbstractChallengeHandler.cancel()WL.Client.AbstractChallen
WL.Client.createProvisioningChallengeHandler()	No replacement. Device provisioning is now handled automatically by the security framework.

Table 5-3. Deprecated JavaScript APIs

API	Migration path

`WLClient.invokeProcedure(WLProcedureInvocationData invocationData,WLResponseListener responseListener)`
 	
`WL.Client.invokeProcedure(invocationData, options)`
 	
`WLClient.invokeProcedure(WLProcedureInvocationData invocationData, WLResponseListener responseListener, WLRequestOptions requestOptions)`
 	
`WLProcedureInvocationResult`
	Use the WLResourceRequest instead.
	Note: The implementation of invokeProcedure uses WLResourceRequest.

`WLClient.getEnvironment`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the device.platform plug-in provides this feature.

Table 5-3. Deprecated JavaScript APIs (continued)

API	Migration path
`WLClient.getLanguage`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the cordova-plugin-globalization plug-in provides this feature.
`WL.Client.connect(options)`	Use WLAuthorizationManager.obtainAccessToken to check connectivity to the server and apply application management rules.

Android APIs

Table 5-4. Discontinued Android API elements

API element	Migration path
`WLConfig WLClient.getConfig()`	No replacement.

`WLDevice WLClient.getWLDevice()`
 	
`WLClient.transmitEvent(org.json.JSONObject event)`
 	
`WLClient.setEventTransmissionPolicy(WLEventTransmissionPolicy policy)`
 	
`WLClient.purgeEventTransmissionBuffer()`
	Use Android API or third-party packages for GeoLocation.

`WL.Client.getUserInfo(realm, key)`
 	
`WL.Client.updateUserInfo(options)`
	No replacement.

`WL.Client.getUserInfo(realm, key`
 	
`WL.Client.updateUserInfo(options)`
	No replacement

`WLClient.checkForNotifications()`	Use WLAuthorizationManager.obtainAccessToken("", listener) to check connectivity to the server and apply application management rules.

`WLClient.login(java.lang.String realmName, WLRequestListener listener, WLRequestOptions options)`
 	
`WLClient.login(java.lang.String realmName, WLRequestListener listener)`
	Use AuthorizationManager.login()

`WLClient.logout(java.lang.String realmName, WLRequestListener listener, WLRequestOptions options)`
 	
`WLClient.logout(java.lang.String realmName, WLRequestListener listener)`
	Use AuthorizationManager.logout().

`WLClient.obtainAccessToken(java.lang.String scope,WLResponseListener responseListener)`	Use WLAuthorizationManager.obtainAccessToken(String, WLAccessTokenListener) to check connectivity to the server and apply application management rules.

`WLClient.getLastAccessToken()`
 	
`WLClient.getLastAccessToken(java.lang.String scope)`
	Use AuthorizationManager.

`WLClient.getRequiredAccessTokenScope(int status, java.lang.String header)`	Use AuthorizationManager.

Table 5-4. Discontinued Android API elements (continued)

API element	**Migration path**
`WLClient.logActivity(java.lang.String activityType)`	Use com.worklight.common.Logger . For more information, see "Logger SDK" on page 11-37.
`WLAuthorizationPersistencePolicy`	No replacement. To implement authorization persistence, store the authorization token in the application code and create custom HTTP requests. For more information, see "Java custom resource-request implementation sample" on page 7-309.

`WLSimpleSharedData.setSharedToken(myName, myValue)`
 	
`WLSimpleSharedData.getSharedToken(myName)`
 	
`WLSimpleSharedData.clearSharedToken(myName)`
	Use the Android APIs to share tokens across applications.

`WLUserCertificateManager.deleteCertificate(android.content.Context context)`	No replacement
BaseChallengeHandler.submitFailure(WLResponse wlResponse)	Use BaseChallengeHandler.cancel().
ChallengeHandler	For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
WLChallengeHandler	Use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
ChallengeHandler.isCustomResponse()	Use GatewayChallengeHandler.canHandleResponse().
ChallengeHandler.submitAdapterAuthentication	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler. For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.

Table 5-5. Deprecated Android API elements

API	**Migration path**
`WLClient.invokeProcedure(WLProcedureInvocationData invocationData, WLResponseListener responseListener)`	Deprecated. Use WLResourceRequest **Note:** The implementation of invokeProcedure uses WLResourceRequest.

`WLClient.connect(WLResponseListener responseListener)`
 	
`WLClient.connect(WLResponseListener responseListener,WLRequestOptions options)`
	Use WLAuthorizationManager.obtainAccessToken("", listener) to check connectivity to the server and apply application management rules.

Table 5-6. Android APIs depending on the legacy `org.apach.http` APIs are no longer supported

API element	Migration path
`org.apache.http.Header[]` is now deprecated. Therefore, the following methods are removed:	
`org.apache.http.Header[] WLResourceRequest.getAllHeaders()`	Use instead the new Map<String, List<String>> WLResourceRequest.getAllHeaders() API.
`WLResourceRequest.addHeader(org.apache.http.Header header)`	Use instead the new WLResourceRequest.addHeader(String name, String value) API.
`org.apache.http.Header[] WLResourceRequest.getHeaders(java.lang.String headerName)`	Use instead the new List<String> WLResourceRequest.getHeaders(String headerName) API.
`org.apache.http.Header WLResourceRequest.getFirstHeader(java.lang.String headerName)`	Use instead the new WLResourceRequest.getHeaders(String headerName) API.
`WLResourceRequest.setHeaders(org.apache.http.Header[] headers)`	Instead, use the new WLResourceRequest.setHeaders(Map<String, List<String>> headerMap) API.
`WLResourceRequest.setHeader(org.apache.http.Header header)`	Instead, use the new WLResourceRequest.setHeaders(Map<String, List<String>> headerMap) API.
`org.apache.http.client.CookieStore WLClient.getCookieStore()`	Replaced with java.net.CookieStore getCookieStore WLClient.getCookieStore() `java.net.CookieStore getCookieStore WLClient.getCookieStore()`
`WLClient.setAllowHTTPClientCircularRedirect(boolean isSet)`	No replacement. MFP Client allows circular redirects.

`WLHttpResponseListenerWLResourceRequest`, all methods that take `WLHttpResponseListener`:
 	
`WLResourceRequest.send(java.util.HashMap formParameters,WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(org.json.JSONObject json, WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(byte[] data, WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(java.lang.String requestBody,WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(WLHttpResponseListener listener)`
 	
`WLClient.sendRequest(org.apache.http.client.methods. HttpUriRequest request,WLHttpResponseListener listener)`
 	
`WLClient.sendRequest(org.apache.http.client.methods. HttpUriRequest request, WLResponseListener listener)`
	Removed due to deprecated Apache HTTP Client dependencies. Create your own request to have full control over the request and response.

The com.worklight.androidgap.api package provides the Android platform functionality for Cordova apps. In MobileFirst, a number of changes were made to accommodate the Cordova integration.

Table 5-7. Discontinued Android gap elements (com.worklight.androidgap.api)

API element	Migration path
The Android activity was replaced with the Android context.	
`static WL.createInstance(android.app.Activity activity)`	`static WL.createInstance(android.content.Context context)` Creates a shared instance.
`static WL.getInstance()`	`static WL.getInstance()` Gets an instance of the `WL` class. This method cannot be called before `WL.createInstance(Context)`.

Objective C API

Table 5-8. Discontinued iOS Objective C APIs

API element	Migration path

`[WLClient getWLDevice][WLClient transmitEvent:]`
 	
`[WLClient setEventTransmissionPolicy]`
 	
`[WLClient purgeEventTransmissionBuffer]`
	Geolocation removed. Use native iOS or third-party packages for GeoLocation.

`WL.Client.getUserInfo(realm, key)`
 	
`WL.Client.updateUserInfo(options)`
	No replacement.

`WL.Client.deleteUserPref(key, options)`	No replacement. You can use an adapter and the MFP.Server.getAuthenticatedUser API to manage user preferences.
`[WLClient getRequiredAccessTokenScopeFromStatus]`	Use WLAuthorizationManager obtainAccessTokenForScope.
`[WLClient login:withDelegate:]`	Use WLAuthorizationManager login.
`[WLClient logout:withDelegate:]`	Use WLAuthorizationManager logout.

`[WLClient lastAccessToken]`
 	
`[WLClient lastAccessTokenForScope:]`
	Use WLAuthorizationManager obtainAccessTokenForScope.

`[WLClient obtainAccessTokenForScope:withDelegate:]`
 	
`[WLClient getRequiredAccessTokenScopeFromStatus: authenticationHeader:]`
	Use WLAuthorizationManager obtainAccessTokenForScope.

`[WLClient isSubscribedToAdapter:(NSString *) adaptereventSource:(NSString *) eventSource`	Use “Objective-C client-side push API for iOS apps” on page 8-5 from the IBMMobileFirstPlatformFoundationPush framework. For more information, see “Migrating to push notifications from event source-based notifications” on page 5-54.

Table 5-8. Discontinued iOS Objective C APIs (continued)

API element	Migration path
`[WLClient - (int) getEventSourceIDFromUserInfo: (NSDictionary *) userInfo]`	Use “Objective-C client-side push API for iOS apps” on page 8-5 from the IBMMobileFirstPlatformFoundationPush framework. For more information, see “Migrating to push notifications from event source-based notifications” on page 5-54.
`[WLClient invokeProcedure: (WLProcedureInvocationData *) ]`	Deprecated. Use WLResourceRequest instead.
`[WLClient sendUrlRequest:delegate:]`	Use [WLResourceRequest sendWithDelegate:delegate] instead.
`[WLClient (void) logActivity:(NSString *) activityType]`	Removed. Use an Objective C logger.

`[WLSimpleDataSharing setSharedToken: myName value: myValue]`
 	
`[WLSimpleDataSharing getSharedToken: myName]]`
 	
`[WLSimpleDataSharing clearSharedToken: myName]`
	Use the OS APIs to share tokens across applications.

BaseChallengeHandler.submitFailure(WLResponse *)challenge	Use BaseChallengeHandler.cancel().
BaseProvisioningChallengeHandler	No replacement. Device provisioning is now handled automatically by the security framework.
ChallengeHandler	For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see “Client security APIs” on page 7-305.
WLChallengeHandler	Use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see “Client security APIs” on page 7-305.
ChallengeHandler.isCustomResponse()	Use GatewayChallengeHandler.canHandleResponse().
ChallengeHandler.submitAdapterAuthentication	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler. For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.

Windows C# API

Table 5-9. Deprecated Windows C# API elements.

Category	Description	Recommended action

Table 5-9. Deprecated Windows C# API elements (continued).

Category	Description	Recommended action
C# API Classes	ChallengeHandler	For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
	ChallengeHandler. isCustomResponse()	Use GatewayChallengeHandler.canHandleResponse().
	ChallengeHandler.submitAdapterAuthentication	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler. For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.
	ChallengeHandler.submitFailure(WLResponse wlResponse)	For custom gateway challenge handlers, use GatewayChallengeHandler.Shouldcancel(). For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.ShouldCancel().
	WLAuthorizationManager	Use WorklightClient.WorklightAuthorizationManager instead.
	WLChallengeHandler	Use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
	WLChallengeHandler.submitFailure(WLResponse wlResponse)	Use SecurityCheckChallengeHandler.ShouldCancel().
	WLClient	Use WorklightClient instead.
	WLErrorCode	Not supported.
	WLFailResponse	Use WorklightResponse instead.
	WLResponse	
	WLProcedureInvocationData	Use WorklightProcedureInvocationData instead.
	WLProcedureInvocationFailResponse	Not supported.
	WLProcedureInvocationResult	Not supported.
	WLRequestOptions	Not supported.
	WLResourceRequest	Use WorklightResourceRequest instead.
C# API Interfaces	WLHttpResponseListener	Not supported.
	WLResponseListener	The response will be available as a WorklightResponse object
	WLAuthorizationPersistencePolicy	Not supported.

Server-side API changes in V8.0.0

To migrate the server side of your MobileFirst application, take into account the changes to the APIs.

The following tables list the discontinued server-side API elements in V8.0.0, deprecated server-side API elements in V8.0.0, and suggested migration paths. For more information about migrating the server side of your application, see

"Migrating existing adapters to work under MobileFirst Server V8.0.0" on page 5-51 and "Migrating to push notifications from event source-based notifications" on page 5-54.

Table 5-10. JavaScript API elements discontinued in V8.0.0.

Category	API Element	Replacement path
Security	WL.Server.getActiveUser	Use MFP.Server.getAuthenticatedUser instead.
	WL.Server.getCurrentUserIdentity	
	WL.Server.getCurrentDeviceIdentity	
	WL.Server.setActiveUser	
	WL.Server.getClientId	
	WL.Server.getClientDeviceContext	
	WL.Server.setApplicationContext	
Event source	WL.Server.createEventSource	To migrate from Event source-based notifications to tag-based notifications, see "Migrating to push notifications from event source-based notifications" on page 5-54.
	WL.Server.setEventHandlers	
	WL.Server.createEventHandler	
	WL.Server.createSMSEventHandler	To send SMS messages, use the push service REST API. For more information, see "Sending SMS notifications" on page 7-264.
	WL.Server.createUSSDEventHandler	Integrate USSD by using third-party services.
Push	WL.Server.getUserNotificationSubscription	To migrate from Event source-based notifications to tag-based notifications, see "Migrating to push notifications from event source-based notifications" on page 5-54.
	WL.Server.notifyAllDevices	
	WL.Server.sendMessage	
	WL.Server.notifyDevice	
	WL.Server.notifyDeviceSubscription	
	WL.Server.notifyAll	
	WL.Server.createDefaultNotification	
	WL.Server.submitNotification	
	WL.Server.subscribeSMS	Use the REST API Push Device Registration (POST) to register the device. To send and receive SMS notifications, provide the **`phoneNumber`** in the payload while invoking the API.
	WL.Server.unsubscribeSMS	Use the REST API Push Device Registration (DELETE) to unregister the device.
	WL.Server.getSMSSubscription	Use the REST API Push Device Registration GET) to get the device registrations.
Location services	WL.Geo.*	Integrate Location services by using third-party services.

Table 5-10. JavaScript API elements discontinued in V8.0.0 (continued).

Category	API Element	Replacement path
WS-Security	WL.Server.signSoapMessage	Use the WS-Security capabilities of WebSphere Application Server.

Table 5-11. Java API elements discontinued in V8.0.0.

Category	API Element	Replacement path
Security	SecurityAPI.getSecurityContext	Use AdapterSecurityContext instead.
Push	PushAPI.sendMessage(INotification notification, String applicationId)	To migrate from Event source-based notifications to tag-based notifications, see "Migrating to push notifications from event source-based notifications" on page 5-54.
	INotification PushAPI.buildNotification();	To migrate from Event source-based notifications to tag-based notifications, see "Migrating to push notifications from event source-based notifications" on page 5-54.
	UserSubscription PushAPI.getUserSubscription(String eventSource, String userId)	To migrate from Event source-based notifications to tag-based notifications, see "Migrating to push notifications from event source-based notifications" on page 5-54.

Table 5-12. Java API elements deprecated in V8.0.0.

Category	Deprecation	Replacement path
Java	AdaptersAPI interface in the com.worklight.adapters.rest.api package	Use the AdaptersAPI interface in the com.ibm.mfp.adapter.api package instead.
	AnalyticsAPI interface in the com.worklight.adapters.rest.api package	Use the AnalyticsAPI interface in the com.ibm.mfp.adapter.api package instead.
	ConfigurationAPI interface in the com.worklight.adapters.rest.api package	Use the ConfigurationAPI interface in the com.ibm.mfp.adapter.api package instead.
	OAuthSecurity annotation in the com.worklight.core.auth package	Use the OAuthSecurity annotation in the com.ibm.mfp.adapter.api package instead.
	MFPJAXRSApplication class in the com.worklight.wink.extensions package	Use the MFPJAXRSApplication class in the com.ibm.mfp.adapter.api package instead.
	WLServerAPI interface in the com.worklight.adapters.rest.api package	Use the JAX-RS Context annotation to access the MobileFirst API interfaces directly.
	WLServerAPIProvider class in the com.worklight.adapters.rest.api package	Use the JAX-RS Context annotation to access the MobileFirst API interfaces directly.

Migrating client applications to IBM MobileFirst Platform Foundation V8.0.0

Migrate your existing client applications to IBM MobileFirst Platform Foundation V8.0.0.

Read the following topics to learn how to migrate your client application from IBM MobileFirst Platform Foundation V7.1 to V8.0.0. To learn more about changes in the development process, in the MobileFirst security framework and push service, and more, see “Migrating apps from earlier releases” on page 5-1.

Migrating existing native iOS applications

To migrate an existing native iOS project that was created with IBM MobileFirst Platform Foundation version 6.2.0 or later, you must modify the project to use the SDK from the current version. Then you replace the client-side APIs that are discontinued or not in V8.0.0. The migration assistance tool can scan your code and generate reports of the APIs to replace.

Scanning existing MobileFirst native iOS apps to prepare for MobileFirst version 8.0:

The migration assistance tool helps you prepare your apps that were created with previous versions of IBM MobileFirst Platform Foundation for migration by scanning the sources of the native iOS apps that were developed by using Swift or Objective-C and generating a report of APIs that are deprecated or discontinued in version 8.0.

Before you begin

The following information is important to know before you use the migration assistance tool:

- You must have an existing IBM MobileFirst Platform Foundation native iOS application.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.
- Review and understand the limitations of the migration process. For more information, see "Migrating apps from earlier releases" on page 5-1.

About this task

Apps that were created with earlier versions of IBM MobileFirst Platform Foundation are not supported in IBM MobileFirst Platform Foundation version 8.0 without some changes. The migration assistance tool simplifies the process by scanning the source files in the existing version app and identifies APIs that are deprecated, no longer supported, or modified in version 8.0.

The migration assistance tool does not modify or move any developer code or comments of your app.

Procedure

1. Download the migration assistance tool by using one of the following methods:
   - Download the `.tgz` file from the Jazzhub repository.
   - Download the Developer Kit, which contains the migration assistance tool as a file named `mfpmigrate-cli.tgz`, from the Download page. For more information about the Developer Kit, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.
   - Download the tool by using the instructions that are provided in "Opening the MobileFirst Operations Console" on page 7-12.
2. Install the migration assistance tool.
   a. Change to the directory where you downloaded the tool.
   b. Use NPM to install the tool by entering the following command:
      ```
      npm install -g
      ```
3. Scan the IBM MobileFirst Platform Foundation app by entering the following command:
   ```
   mfpmigrate scan --in source_directory --out destination_directory
   --type ios
   ```
   source_directory
   The current location of the version project.

   destination_directory
   The directory where the report is created.

   When it is used with the scan command, the migration assistance tool identifies APIs in the existing IBM MobileFirst Platform Foundation app that are removed, deprecated, or changed in version 8.0 and saves them in the identified destination directory.

Migrating an existing iOS project to version V8.0.0 manually:

Migrate your existing native iOS project manually within your Xcode project and continue developing with IBM MobileFirst Platform Foundation V8.0.0.

Before you begin

Before you begin you must:
- be working in Xcode 7.0 (iOS 9) or later.
- have an existing native iOS project that was created with IBM MobileFirst Platform Foundation 6.2.0 or later.
- have access to a copy of the V8.0.0 MobileFirst iOS SDK files. See "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26.

Procedure
1. Delete all of the existing references to the static library `libWorklightStaticLibProjectNative.a` in the **Link Binary With Libraries** tab of **Build Phases** section.
2. Delete the `Headers` folder from the `WorklightAPI` folder.
3. In the **Build Phases** section, link the main required framework `IBMMobileFirstPlatformFoundation.framework` file in the **Link Binary With Libraries** tab.

   This framework provides core MobileFirst functionality. Similarly, you can add other frameworks for optional functionality (see Table 7-6 on page 7-32).
4. Similar to the preceding step, link the following resources to your project in the **Link Binary With Libraries** section of the **Build Phases** tab.
   - `SystemConfiguration.framework`
   - `MobileCoreServices.framework`
   - `Security.framework`
   -

     Note: Some frameworks might already be linked.
   - `libstdc++.6.tbd`
   - `libz.tbd`
   - `libc++.tbd`
5. Remove `$(SRCROOT)/WorklightAPI/include` from the header search path.
6. Replace all of the existing MobileFirst imports of headers with a single entry of the following new umbrella header:
   - Objective C:
     ```
     #import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
     ```
   - Swift:
     ```
     import IBMMobileFirstPlatformFoundation
     ```

Results

Your application is now upgraded to work with the IBM MobileFirst Platform Foundation, V8.0.0 iOS SDK.

What to do next
- Replace the client-side APIs that are discontinued or not in V8.0.0. For more information about the changes in the client-side API, see "Updating the iOS

code" on page 5-23. For more information about the migration assistance tool, see "Scanning existing MobileFirst native iOS apps to prepare for MobileFirst version 8.0" on page 5-17.
- Before you can access server resources, you must register your app. See "Registering iOS applications to MobileFirst Server" on page 7-37. For details about the `mfpclient.plist` file, see "iOS client properties file" on page 7-41.

Migrating an existing native iOS projects to version V8.0.0 with CocoaPods:

Migrate your existing native iOS project to work with V8.0.0 by getting the IBM MobileFirst Platform Foundation iOS SDK using CocoaPods and making changes in the project configuration.

Before you begin

Note: MobileFirst development is supported in Xcode from version 7.1 by using iOS 8.0 and later.

You must have:
- CocoaPods installed in your development environment. For more information, see the "Getting Started" guide for CocoaPods installation.
- Xcode 7.1 with iOS 8.0 or higher for your development environment.
- An app integrated with MobileFirst 6.2 or later.

About this task

The SDK contains required and optional SDKs. Each required or optional SDK has its own pod.

The required `IBMMobileFirstPlatformFoundation` pod is the core of the system. It implements client-to-server connections, handles security, analytics, and application management.

The following optional pods provide additional features.

Table 5-13. Pods for installing optional frameworks

Pod	Feature
`IBMMobileFirstPlatformFoundationPush`	Adds the `IBMMobileFirstPlatformFoundationPush` framework for enabling Push. For more information, see "Push notification" on page 7-248.
`IBMMobileFirstPlatformFoundationJSONStore`	Implements the JSONStore feature. Include this pod in your Podfile if you intend to use the JSONStore feature in your app. See "JSONStore" on page 7-134.
`IBMMobileFirstPlatformFoundationOpenSSLUtils`	Contains the MobileFirst embedded OpenSSL feature and loads automatically the `openssl` framework. Include this pod in your Podfile if you intend to use the OpenSSL provided by MobileFirst. For more information on OpenSSL options, see "Enabling OpenSSL for iOS" on page 7-47.

Procedure

1. Open your project in Xcode.
2. Delete the `WorklightAPI` folder from your Xcode project (move it to trash).
3. Modify your existing code in the following ways:
   a. Remove `$(SRCROOT)/WorklightAPI/include` from the header search path.
   b. Remove `$(PROJECTDIR)/WorklightAPI/frameworks` from the frameworks search path.
   c. Remove any references to the static library`libWorklightStaticLibProjectNative.a`.
4. In the **Build Phases** tab, remove the links to the following frameworks and libraries (these are re-added automatically by CocoaPods):
   - `libWorklightStaticLibProjectNative.a`
   - `SystemConfiguration.framework`
   - `MobileCoreServices.framework`
   - `CoreData.framework`
   - `CoreLocation.framework`
   - `Security.framework`
   - `sqlcipher.framework`
   - `libstdc++.6.dylib`
   - `libz.dylib`
5. Close Xcode.
6. 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 and save the changes, including the iOS version:
```
use_frameworks!
platform :ios, 9.0
pod 'IBMMobileFirstPlatformFoundation'
```
   f. Specify additional pods in the file from the list above, if your app needs to use the additional functionality that they provide. For example, if your app uses OpenSSL, the `Podfile` might look like this:
```
use_frameworks!
platform :ios, 9.0
pod 'IBMMobileFirstPlatformFoundation'
pod 'IBMMobileFirstPlatformFoundationOpenSSLUtils'
```
   For a list of optional pods, see Table 7-7 on page 7-34.

   Note:

   The previous syntax imports the latest version of the `IBMMobileFirstPlatformFoundation` pod. If you are not using the latest version of MobileFirst, you need to add the full version number, including the major, minor, and patch numbers. The patch number is in the format *YYYYMMDDHH*. For example, for importing the specific patch version 8.0.2016021411 of the `IBMMobileFirstPlatformFoundation` pod the line would look like this:
```
pod 'IBMMobileFirstPlatformFoundation', '8.0.2016021411'
```

Or to get the last patch for the minor version number the syntax such is

```
pod 'IBMMobileFirstPlatformFoundation', '~>8.0.0'
```

g. Verify that the Xcode project is closed.

h. 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.

7. 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.

8. Replace all of the existing MobileFirst imports of headers with a single entry of the following new umbrella header:

   - Objective C:

   ```
   #import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
   ```

   - Swift:

   ```
   import IBMMobileFirstPlatformFoundation
   ```

   If you are using Push or JSONStore, you need to include an independent import.

   Push

   - For Objective C:

   ```
   #import
   <IBMMobileFirstPlatformFoundationPush/IBMMobileFirstPlatformFoundationPush.h>
   ```

   - For Swift:

   ```
   import IBMMobileFirstPlatformFoundationPush
   ```

   JSONStore

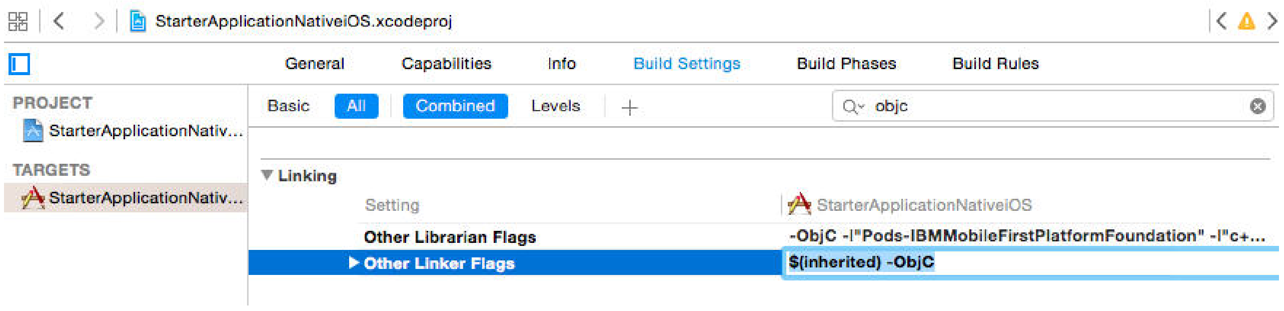
   - For Objective C:

   ```
   #import <IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJS
   ```

   - For Swift:

   ```
   import IBMMobileFirstPlatformFoundationJSONStore
   ```

9. In the **Build Settings** tab, under **Other Linker Flags**, add `$(inherited)` at the beginning of the -ObjC flag. For example:


StarterApplicationNativeiOS.xcodeproj
General
Capabilities
Info
Build Settings
Build Phases
Build Rules
PROJECT
StarterApplicationNativ...
TARGETS
StarterApplicationNativ...
Basic
All
Combined
Levels
objc
Linking
Setting
StarterApplicationNativeiOS

Other Librarian Flags -ObjC -l"Pods-IBMMobileFirstPlatformFoundation" -l"c+...

Other Linker Flags $(inherited) -ObjC

Figure 5-1. Adding $(inherited) to ObjC flag in Xcode Build Settings

10. Beginning with Xcode 7, TLS must be enforced, see "Enforcing TLS-secure connections in iOS apps" on page 7-46.

Results

Your application is now upgraded to work with the IBM MobileFirst Platform Foundation, V8.0.0 iOS SDK.

What to do next

- Replace the client-side APIs that are discontinued or not in V8.0.0. For more information about the changes in the client-side API, see "Updating the iOS code." For more information about the migration assistance tool, see "Scanning existing MobileFirst native iOS apps to prepare for MobileFirst version 8.0" on page 5-17.
- Before you can access server resources, you must register your app. See "Registering iOS applications to MobileFirst Server" on page 7-37. For details about the `mfpclient.plist` file, see "iOS client properties file" on page 7-41.

Migrating encryption in iOS:

If your iOS application used OpenSSL encryption, you might want to migrate your app to the new V8.0.0 native encryption. Also, if you want to continue using OpenSSL, you must install some additional frameworks.

For more information on the iOS encryption options for migration, see "Enabling OpenSSL for iOS" on page 7-47.

Updating the iOS code:

After updating the iOS framework and making necessary configuration changes, a number of issues can be relevant to your specific application code.

The iOS API changes are listed in the Table 1.

For some sample code for creating the client and accessing the server with the new V8.0.0 client for iOS, see "Creating some initial code in iOS" on page 7-42.

Table 5-14. Discontinued iOS Objective C APIs

API element	Migration path

`[WLClient getWLDevice][WLClient transmitEvent:]`
 	
`[WLClient setEventTransmissionPolicy]`
 	
`[WLClient purgeEventTransmissionBuffer]`
	Geolocation removed. Use native iOS or third-party packages for GeoLocation.

`WL.Client.getUserInfo(realm, key)`
 	
`WL.Client.updateUserInfo(options)`
	No replacement.

`WL.Client.deleteUserPref(key, options)`	No replacement. You can use an adapter and the MFP.Server.getAuthenticatedUser API to manage user preferences.
`[WLClient getRequiredAccessTokenScopeFromStatus]`	Use WLAuthorizationManager obtainAccessTokenForScope.
`[WLClient login:withDelegate:]`	Use WLAuthorizationManager login.
`[WLClient logout:withDelegate:]`	Use WLAuthorizationManager logout.

`[WLClient lastAccessToken]`
 	
`[WLClient lastAccessTokenForScope:]`
	Use WLAuthorizationManager obtainAccessTokenForScope.

Table 5-14. Discontinued iOS Objective C APIs (continued)

API element	Migration path

`[WLClient obtainAccessTokenForScope:withDelegate:]`
 	
`[WLClient getRequiredAccessTokenScopeFromStatus: authenticationHeader:]`
	Use WLAuthorizationManager obtainAccessTokenForScope.

`[WLClient isSubscribedToAdapter:(NSString *) adaptereventSource:(NSString *) eventSource`	Use "Objective-C client-side push API for iOS apps" on page 8-5 from the IBMMobileFirstPlatformFoundationPush framework. For more information, see "Migrating to push notifications from event source-based notifications" on page 5-54.
`[WLClient - (int) getEventSourceIDFromUserInfo: (NSDictionary *) userInfo]`	Use "Objective-C client-side push API for iOS apps" on page 8-5 from the IBMMobileFirstPlatformFoundationPush framework. For more information, see "Migrating to push notifications from event source-based notifications" on page 5-54.
`[WLClient invokeProcedure: (WLProcedureInvocationData *) ]`	Deprecated. Use WLResourceRequest instead.
`[WLClient sendUrlRequest:delegate:]`	Use [WLResourceRequest sendWithDelegate:delegate] instead.
`[WLClient (void) logActivity:(NSString *) activityType]`	Removed. Use an Objective C logger.

`[WLSimpleDataSharing setSharedToken: myName value: myValue]`
 	
`[WLSimpleDataSharing getSharedToken: myName]]`
 	
`[WLSimpleDataSharing clearSharedToken: myName]`
	Use the OS APIs to share tokens across applications.

BaseChallengeHandler.submitFailure(WLResponse *)challenge	Use BaseChallengeHandler.cancel().
BaseProvisioningChallengeHandler	No replacement. Device provisioning is now handled automatically by the security framework.
ChallengeHandler	For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
WLChallengeHandler	Use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
ChallengeHandler.isCustomResponse()	Use GatewayChallengeHandler.canHandleResponse().
ChallengeHandler.submitAdapterAuthentication	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler. For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.

Migrating existing native Android applications

To migrate an existing native Android project that was created with IBM MobileFirst Platform Foundation version 6.2.0 or later, you must modify the project to use the SDK from the current version. Then you replace the client-side APIs that are discontinued or not in V8.0.0. The migration assistance tool can scan your code and generate reports of the APIs to replace.

Scanning existing MobileFirst native Android apps to prepare for MobileFirst version 8.0:

The migration assistance tool helps you prepare your apps that were created with a previous version of IBM MobileFirst Platform Foundation for migration by scanning the sources of the native Android app and generating a report of APIs that are deprecated or discontinued in version 8.0.

Before you begin

The following information is important to know before you use the migration assistance tool:

- You must have an existing IBM MobileFirst Platform Foundation native Android application.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.
- Review and understand the limitations of the migration process. For more information, see “Migrating apps from earlier releases” on page 5-1.

About this task

Apps that were created with previous versions of IBM MobileFirst Platform Foundation are not supported in version 8.0 without some changes. The migration assistance tool simplifies the process by scanning the source files in the existing app and identifies APIs that are deprecated, no longer supported, or modified in version 8.0.

The migration assistance tool does not modify or move any developer code or comments of your app.

Procedure

1. Download the migration assistance tool by using one of the following methods:
   - Download the `.tgz` file from the Jazzhub repository.
   - Download the Developer Kit, which contains the migration assistance tool as a file named `mfpmigrate-cli.tgz`, from the Download page. For more information about the Developer Kit, see “The IBM MobileFirst Platform Foundation Developer Kit” on page 7-9.
   - Download the tool by using the instructions that are provided in “Opening the MobileFirst Operations Console” on page 7-12.
2. Install the migration assistance tool.
   a. Change to the directory where you downloaded the tool.
   b. Use NPM to install the tool by entering the following command:
   ```
   npm install -g
   ```
3. Scan the IBM MobileFirst Platform Foundation app by entering the following command:
   ```
   mfpmigrate scan --in source_directory --out destination_directory
   --type android
   ```
   source_directory
   The current location of the project.

   destination_directory
   The directory where the report is created.

When it is used with the scan command, the migration assistance tool identifies APIs in the existing IBM MobileFirst Platform Foundation app that are removed, deprecated, or changed in version 8.0 and saves them in the identified destination directory.

Migrating an Android project with Gradle:

Migrate your Android application with MobileFirst SDK using Gradle.

Before you begin

Ensure that your Android Studio and the Android SDK are set up properly. For more information about how to set up your system, see Android Studio Overview. Your project must conform to the Android Studio/Gradle setup and compile without errors before you upgrade to IBM MobileFirst Platform Foundation.

Note: This task assumes that the Android project is created with Android Studio and that the MobileFirst SDK is added with as described in Adding the IBM MobileFirst Platform Foundation SDK to a new or existing application with Android Studio (7.1). If you need to create a new project, see "Methods of setting up your environment" on page 7-53 instead.

About this task

If your Android Studio project was set up to add a previous version of MobileFirst SDK, remove the `compile` group from the `build.gradle` dependencies enclosure. For example, if you are upgrading from 7.1, remove this group:

```
compile group: 'com.ibm.mobile.foundation',
          name:'ibmmobilefirstplatformfoundation',
          version:'7.1.0.0',
          ext: 'aar',
          transitive: true
```

You can now add the V8.0.0 SDK and configuration, by using local or remote SDK files. See "Setting up Android Studio projects with Gradle" on page 7-53.

Note: After you import the new SDK, you need to import the Javadoc files manually. See "Registering Javadocs to an Android Studio Gradle project" on page 7-57.

Results

You can now start developing your native Android application with the MobileFirst SDK. You might need to adapt your code to changes in the V8.0.0 API (see "Updating the Android code" on page 5-27).

What to do next

- Replace the client-side APIs that are discontinued or not in V8.0.0. For more information about the changes in the client-side API, see "Updating the Android code" on page 5-27. For more information about the migration assistance tool, see "Scanning existing MobileFirst native Android apps to prepare for MobileFirst version 8.0" on page 5-25.
- Before you can access server resources, you must register your app. See "Registering Android applications to MobileFirst Server" on page 7-59. For information about the `mfpclient.plist` file, see "Android client properties file" on page 7-62.

Updating the Android code:

MobileFirstV8.0.0 introduces a number of changes to the Android SDK that might require changes to apps developed in earlier versions.

The tables below list changes in the MobileFirst Android SDK.

For some sample code for creating the client and accessing the server with the new V8.0.0 client for Android, see "Some initial code for accessing the server" on page 7-63.

Table 5-15. Discontinued Android API elements

API element	Migration path
`WLConfig WLClient.getConfig()`	No replacement.

`WLDevice WLClient.getWLDevice()`
 	
`WLClient.transmitEvent(org.json.JSONObject event)`
 	
`WLClient.setEventTransmissionPolicy(WLEventTransmissionPolicy policy)`
 	
`WLClient.purgeEventTransmissionBuffer()`
	Use Android API or third-party packages for GeoLocation.

`WL.Client.getUserInfo(realm, key)`
 	
`WL.Client.updateUserInfo(options)`
	No replacement.

`WL.Client.getUserInfo(realm, key`
 	
`WL.Client.updateUserInfo(options)`
	No replacement

`WLClient.checkForNotifications()`	Use WLAuthorizationManager.obtainAccessToken("", listener) to check connectivity to the server and apply application management rules.

`WLClient.login(java.lang.String realmName, WLRequestListener listener, WLRequestOptions options)`
 	
`WLClient.login(java.lang.String realmName, WLRequestListener listener)`
	Use AuthorizationManager.login()

`WLClient.logout(java.lang.String realmName, WLRequestListener listener, WLRequestOptions options)`
 	
`WLClient.logout(java.lang.String realmName, WLRequestListener listener)`
	Use AuthorizationManager.logout().

`WLClient.obtainAccessToken(java.lang.String scope,WLResponseListener responseListener)`	Use WLAuthorizationManager.obtainAccessToken(String, WLAccessTokenListener) to check connectivity to the server and apply application management rules.

`WLClient.getLastAccessToken()`
 	
`WLClient.getLastAccessToken(java.lang.String scope)`
	Use AuthorizationManager.

`WLClient.getRequiredAccessTokenScope(int status, java.lang.String header)`	Use AuthorizationManager.
`WLClient.logActivity(java.lang.String activityType)`	Use com.worklight.common.Logger . For more information, see "Logger SDK" on page 11-37.

Table 5-15. Discontinued Android API elements (continued)

API element	Migration path
`WLAuthorizationPersistencePolicy`	No replacement. To implement authorization persistence, store the authorization token in the application code and create custom HTTP requests. For more information, see "Java custom resource-request implementation sample" on page 7-309.

`WLSimpleSharedData.setSharedToken(myName, myValue)`
 	
`WLSimpleSharedData.getSharedToken(myName)`
 	
`WLSimpleSharedData.clearSharedToken(myName)`
	Use the Android APIs to share tokens across applications.

`WLUserCertificateManager.deleteCertificate(android.content.Context context)`	No replacement
BaseChallengeHandler.submitFailure(WLResponse wlResponse)	Use BaseChallengeHandler.cancel().
ChallengeHandler	For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
WLChallengeHandler	Use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
ChallengeHandler.isCustomResponse()	Use GatewayChallengeHandler.canHandleResponse().
ChallengeHandler.submitAdapterAuthentication	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler. For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.

Table 5-16. Android APIs depending on the legacy `org.apach.http` APIs are no longer supported

API element	Migration path
`org.apache.http.Header[]` is now deprecated. Therefore, the following methods are removed:	
`org.apache.http.Header[] WLResourceRequest.getAllHeaders()`	Use instead the new Map<String, List<String>> WLResourceRequest.getAllHeaders() API.
`WLResourceRequest.addHeader(org.apache.http.Header header)`	Use instead the new WLResourceRequest.addHeader(String name, String value) API.

Table 5-16. Android APIs depending on the legacy `org.apach.http` APIs are no longer supported (continued)

API element	Migration path
`org.apache.http.Header[] WLResourceRequest.getHeaders(java.lang.String headerName)`	Use instead the new List<String> WLResourceRequest.getHeaders(String headerName) API.
`org.apache.http.Header WLResourceRequest.getFirstHeader(java.lang.String headerName)`	Use instead the new WLResourceRequest.getHeaders(String headerName) API.
`WLResourceRequest.setHeaders(org.apache.http.Header[] headers)`	Instead, use the new WLResourceRequest.setHeaders(Map<String, List<String>> headerMap) API.
`WLResourceRequest.setHeader(org.apache.http.Header header)`	Instead, use the new WLResourceRequest.setHeaders(Map<String, List<String>> headerMap) API.
`org.apache.http.client.CookieStore WLClient.getCookieStore()`	Replaced with java.net.CookieStore getCookieStore WLClient.getCookieStore() `java.net.CookieStore getCookieStore WLClient.getCookieStore()`
`WLClient.setAllowHTTPClientCircularRedirect(boolean isSet)`	No replacement. MFP Client allows circular redirects.

`WLHttpResponseListenerWLResourceRequest`, all methods that take `WLHttpResponseListener`:
 	
`WLResourceRequest.send(java.util.HashMap formParameters,WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(org.json.JSONObject json, WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(byte[] data, WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(java.lang.String requestBody,WLHttpResponseListener listener)`
 	
`WLResourceRequest.send(WLHttpResponseListener listener)`
 	
`WLClient.sendRequest(org.apache.http.client.methods. HttpUriRequest request,WLHttpResponseListener listener)`
 	
`WLClient.sendRequest(org.apache.http.client.methods. HttpUriRequest request, WLResponseListener listener)`
	Removed due to deprecated Apache HTTP Client dependencies. Create your own request to have full control over the request and response.

Migrating existing native Windows applications

To migrate an existing native Windows project that was created with IBM MobileFirst Platform Foundation version 6.2.0 or later, you must modify the project to use the SDK from the current version. Then you replace the client-side APIs that are discontinued or not in V8.0.0. The migration assistance tool can scan your code and generate reports of the APIs to replace.

Scanning existing MobileFirst native Windows apps to prepare for MobileFirst version 8.0:

The migration assistance tool helps you prepare your apps that were created with earlier versions of IBM MobileFirst Platform Foundation for migration by scanning the sources of the native Windows app and generating a report of APIs that are deprecated or discontinued in version 8.0.

Before you begin

The following information is important to know before you use the migration assistance tool:

- You must have an existing IBM MobileFirst Platform Foundation native Windows application.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.
- Review and understand the limitations of the migration process. For more information, see "Migrating apps from earlier releases" on page 5-1.

About this task

Apps that were created with earlier versions of IBM MobileFirst Platform Foundation are not supported in version 8.0 without some changes. The migration assistance tool simplifies the process by scanning the source files in the existing native Windows app and identifies APIs that are deprecated, no longer supported, or modified in version 8.0.

The migration assistance tool does not modify or move any developer code or comments of your app.

Procedure

1. Download the migration assistance tool by using one of the following methods:
   - Download the `.tgz` file from the Jazzhub repository.
   - Download the Developer Kit, which contains the migration assistance tool as a file named `mfpmigrate-cli.tgz`, from the Download page. For more information about the Developer Kit, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.
   - Download the tool by using the instructions that are provided in "Opening the MobileFirst Operations Console" on page 7-12.
2. Install the migration assistance tool.
   a. Change to the directory where you downloaded the tool.
   b. Use NPM to install the tool by entering the following command:
      ```
      npm install -g
      ```
3. Scan the IBM MobileFirst Platform Foundation app by entering the following command:
   ```
   mfpmigrate scan --in source_directory --out destination_directory
   --type windows
   ```
   source_directory
      The current location of the project.

   destination_directory
      The directory where the report is created.

When it is used with the scan command, the migration assistance tool identifies APIs in the existing IBM MobileFirst Platform Foundation app that are removed, deprecated, or changed in version 8.0 and saves them in the identified destination directory.

Migrating a Windows project:

To work with existing native Windows project that was created with IBM MobileFirst Platform Foundation V6.2.0 or later, you must modify the project.

About this task

MobileFirst V8.0.0 only supports Windows Universal environments, that is Windows 10 Universal Windows Platform (UWP) and Windows 8 Universal (Desktop and Phone). Windows Phone 8 Silverlight is not supported.

You can upgrade your Visual Studio project to V8.0.0 manually. MobileFirstV8.0.0 introduces a number of changes to the Visual Studio SDK that may require changes to apps developed in earlier versions. For information on the API's that have changed, see "Updating the Windows code" on page 5-32.

Procedure

1. Update your MobileFirst SDK to V8.0.0.
   a. Remove the MobileFirst SDK packages manually. This includes the `wlclient.properties` file, as well as the following references:
      - `Newtonsoft.Json`
      - `SharpCompress`
      - `worklight-windows8`

      Note: If your app uses the application authenticity or extended authenticity feature, you must add either Microsoft Visual C++ 2013 Runtime Package for Windows or Microsoft Visual C++ 2013 Runtime Package for Windows Phone as a reference to your app. To so do, in Visual Studio, right-click on the references of your native project and complete one of the following choices depending on which environment you added to your native API app:
      - For Windows desktops and tablets: Right click **References** > **Add reference** > **Windows 8.1** > **Extensions** > **Microsoft Visual C++ 2013 Runtime Package for Windows** > **OK**.
      - For Windows Phone 8 Universal: Right click **References** > **Add reference** > **Windows 8.1** > **Extensions** > **Microsoft Visual C++ 2013 Runtime Package for Windows Phone** > **OK**.
      - For Windows 10 Universal Windows Platform (UWP): Right click **References** > **Add reference** > **Windows 8.1** > **Extensions** > **Microsoft Visual C++ 2013 Runtime Package for Windows Universal** > **OK**.

   b. Add the MobileFirst V8.0.0 SDK packages through NuGet. See "Adding the MobileFirst SDK by using NuGet" on page 7-66.
2. Updating your application code to use MobileFirst V8.0.0 API's.
   a. For earlier releases, the Windows API's were part of the IBM.Worklight.namespace. These API's are now obsolete and have been replaced by equivalent WorklightNamespace API in the. You need to modify the app to replace all references to the IBM.Worklight.namespace with the corresponding equivalent in the WorklightNamespace.

For example, the following snippet is an example of using the

```
WLResourceRequest request = new WLResourceRequest
                             (new Uri(uriBuilder.ToString()), "GET", "accessRestricted");
                             request.send(listener);
```

The snippet updated with the new API would be :

```
WorklightResourceRequest request = newClient.ResourceRequest
                             (new Uri(uriBuilder.ToString(), UriKind.Relative), "GET", "accessRestricted");
                             WorklightResponse response = await request.Send();
```

b. All methods that performed asynchronous operations previously used a Response listener call back model. These have been replaced by the await/async model.

Results

You can now start developing your native Windows application with the MobileFirst SDK. You might need to update your code to reflect the changes for MobileFirstV8.0.0 API.

What to do next

- Replace the client-side APIs that are discontinued or not in V8.0.0. For more information about the changes in the client-side API, see “Updating the Windows code.” For more information about the migration assistance tool, see “Scanning existing MobileFirst native Windows apps to prepare for MobileFirst version 8.0” on page 5-30.
- You need to register your app before you can access server resources. See “Registering Windows applications to MobileFirst Server” on page 7-68. Also see “Client property file for Windows 10 Universal Windows Platform and Windows 8 Universal” on page 7-72 for information on the `mfpclient.resw` file.

Updating the Windows code:

MobileFirstV8.0.0 introduces a number of changes to the Windows SDK that might require changes to apps developed in earlier versions.

Table 5-17. Deprecated Windows C# API elements.

Category	Description	Recommended action

Table 5-17. Deprecated Windows C# API elements (continued).

Category	Description	Recommended action
C# API Classes	ChallengeHandler	For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
	ChallengeHandler. isCustomResponse()	Use GatewayChallengeHandler.canHandleResponse().
	ChallengeHandler.submitAdapterAuthentication	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler. For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.
	ChallengeHandler.submitFailure(WLResponse wlResponse)	For custom gateway challenge handlers, use GatewayChallengeHandler.Shouldcancel(). For MobileFirst security-check challenge handlers, use SecurityCheckChallengeHandler.ShouldCancel().
	WLAuthorizationManager	Use WorklightClient.WorklightAuthorizationManager instead.
	WLChallengeHandler	Use SecurityCheckChallengeHandler. For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
	WLChallengeHandler.submitFailure(WLResponse wlResponse)	Use SecurityCheckChallengeHandler.ShouldCancel().
	WLClient	Use WorklightClient instead.
	WLErrorCode	Not supported.
	WLFailResponse	Use WorklightResponse instead.
	WLResponse	
	WLProcedureInvocationData	Use WorklightProcedureInvocationData instead.
	WLProcedureInvocationFailResponse	Not supported.
	WLProcedureInvocationResult	Not supported.
	WLRequestOptions	Not supported.
	WLResourceRequest	Use WorklightResourceRequest instead.
C# API Interfaces	WLHttpResponseListener	Not supported.
	WLResponseListener	The response will be available as a WorklightResponse object
	WLAuthorizationPersistencePolicy	Not supported.

Migrating existing Cordova and hybrid applications

To migrate an existing Cordova or hybrid application that was created with IBM MobileFirst Platform Foundation version 6.2.0 or later, you must create a Cordova project that uses the plug-ins from the current version. Then you replace the client-side APIs that are discontinued or not in V8.0.0. The migration assistance tool can help you in this task.

To migrate a Cordova or hybrid app, you need to

- Update your project to V8.0.0. The migration assistance tool can help you in this task. For more information, see "Migrating existing hybrid or cross-platform apps to Cordova apps supported by MobileFirst version 8.0" on page 5-38.
- Replace the client-side APIs that are discontinued or not in V8.0.0. The migration assistance tool can scan your code and generate reports of the APIs to replace. For a list of API changes, see "Upgrading the WebView" on page 5-45.
- If you use Direct Update, review "Migrating Direct Update" on page 5-44.

Comparison of Cordova apps developed with V8.0.0 versus V7.1.0 and before:

Compare Cordova apps developed with IBM MobileFirst Platform Foundation V8.0.0 and Cordova and hybrid apps developed with IBM MobileFirst Platform Foundation V7.1.

Table 5-18. Cordova developed with IBM MobileFirst Platform Foundation V8.0.0 versus Cordova and hybrid apps developed with IBM MobileFirst Platform Foundation V7.1.

Feature	**Cordova app with IBM MobileFirst Platform Foundation V8.0.0**	**Cordova app with IBM MobileFirst Platform Foundation V7.1**	**MobileFirst hybrid app with IBM MobileFirst Platform Foundation V7.1**
IDE Eclipse Studio			
Eclipse plug-in and integration	Yes. For more information, see "IBM MobileFirst Studio plug-in for managing Cordova projects in Eclipse" on page 7-114.	Unsupported	Yes (Proprietary)
Application Components	Yes (Cordova) **Note:** Create your own Cordova plug-ins to manage application components in your organization. For more information, see Apache Cordova Plugin Development Guide.	Yes (Cordova) **Note:** Create your own Cordova plug-ins to manage application components in your organization. For more information, see Apache Cordova Plugin Development Guide.	Yes (Proprietary)
Project Templates	Yes (Cordova) **Note:** Use the Apache Cordova `cordova create --template` command.	Yes (Cordova) **Note:** Use `mfp cordova create --template` or the Apache Cordova command `cordova create --copy-from`	Yes (Proprietary)
Dojo and jQuery IDE instrumentation	Yes **Note:** Dojo and jQuery Mobile are JavaScript frameworks that you can use in Cordova apps. For more information, see Dojo Documentation and jQuery Mobile documentation.	Yes **Note:** Dojo and jQuery Mobile are JavaScript frameworks that you can use in Cordova apps. For more information, see Dojo Documentation and jQuery Mobile documentation.	Yes
Mobile UI Patterns	Unsupported	Unsupported	Deprecated
Application sub types			

Table 5-18. Cordova developed with IBM MobileFirst Platform Foundation V8.0.0 versus Cordova and hybrid apps developed with IBM MobileFirst Platform Foundation V7.1. (continued)

Feature	**Cordova app with IBM MobileFirst Platform Foundation V8.0.0**	**Cordova app with IBM MobileFirst Platform Foundation V7.1**	**MobileFirst hybrid app with IBM MobileFirst Platform Foundation V7.1**
Shell Component	Unsupported **Note:** If the previous Hybrid app used shells and inner applications, it is recommended to adopt Cordova design patterns and implement the shell components as Cordova plug-ins, that can be shared across applications.	Unsupported	Yes
Inner Hybrid Application	Unsupported **Note:** If the previous Hybrid app used shells and inner applications, it is recommended to adopt Cordova design patterns and implement the shell components as Cordova plug-ins, that can be shared across applications.	Unsupported	Yes
Application Features			
Mobile OS	iOS 8 or higher, Android 4.1 or higher, Windows Phone 8.1, Windows Phone 10. For more information, see “System requirements” on page 2-7.	iOS 7 or higher, Android 4 or higher. For more information, see IBM MobileFirst Platform Foundation V7.1 system requirements.	iOS, Android, and Windows Phone 8
Web applications	Yes, as a JavaScript application developed without Apache Cordova. For more information, see “Developing web applications” on page 7-73.	Unsupported	Yes, as a desktopbrowser or mobilewebapp environment.
Direct Update	Yes. For more information, see “Updating Cordova client apps directly” on page 7-235.	Yes	Yes
MobileFirst Security Framework	Yes	Yes	Yes
Application Authenticity	Yes. For more information, see “Enabling the application-authenticity security check” on page 7-282.	Yes	Yes
Certificate pinning	Yes. For more information, see “Certificate pinning” on page 7-185.	No	Yes
JSONStore	Yes. Use the cordova-plugin-mfp-jsonstore plug-in.	Yes	Yes

Table 5-18. Cordova developed with IBM MobileFirst Platform Foundation V8.0.0 versus Cordova and hybrid apps developed with IBM MobileFirst Platform Foundation V7.1. (continued)

Feature	Cordova app with IBM MobileFirst Platform Foundation V8.0.0	Cordova app with IBM MobileFirst Platform Foundation V7.1	MobileFirst hybrid app with IBM MobileFirst Platform Foundation V7.1
FIPS 140-2	Yes. Use the cordova-plugin-mfp-fips plug-in. **Restriction:** FIPS is supported for Android and iOS. FIPS is not supported for Windows. For more information, see "Enabling FIPS 140-2" on page 10-77.	No	Yes
Encryption of web resources that are associated with the application within the application binary file.	Yes. For more information, see "Encrypting the web resources of your Cordova packages" on page 7-112.	No	Yes
Verification of the integrity of web resources by using a checksum each time the app starts running.	Yes. For more information, see "Enabling the web resources checksum feature" on page 7-113.	Unsupported	Yes
Specification of the app's target category (B2E or B2C) for addressable device license tracking.	Yes. For more information, see "Setting the application license information" on page 10-80.	No	Yes
Simple data sharing	No	Yes	Yes
Single sign-on	Yes **Note:** Device single sign-on (SSO) is now supported by way of the new predefined enableSSO security-check application-descriptor configuration property. For more information, see "Configuring device single sign-on (SSO)" on page 7-301.	Yes	Yes
MobileFirst application skins	No **Note:** To detect and handle different device screen sizes, use standard web development practices such as responsive web design.	No **Note:** To detect and handle different device screen sizes, use standard web development practices such as responsive web design.	Yes
Environment optimizations	Yes (Cordova). Use the `merges` directory to define web resources specific to a platform. For more information, see Using *merges* to Customize Each Platform in the Apache Cordova documentation.	Yes (Cordova). Use the merges directory to define web resources specific to a platform. For more information, see Using *merges* to Customize Each Platform in the Apache Cordova documentation.	Yes (Proprietary)

Table 5-18. Cordova developed with IBM MobileFirst Platform Foundation V8.0.0 versus Cordova and hybrid apps developed with IBM MobileFirst Platform Foundation V7.1. (continued)

Feature	**Cordova app with IBM MobileFirst Platform Foundation V8.0.0**	**Cordova app with IBM MobileFirst Platform Foundation V7.1**	**MobileFirst hybrid app with IBM MobileFirst Platform Foundation V7.1**
Push Notifications	Yes. Use the cordova-plugin-mfp-push plug-in. **Restriction:** You can map predefined MobileFirst security checks only to the push.mobileclient scope. Custom security checks are not supported because JavaScript challenge handlers are not called.	Yes **Note:** For Android, you must add the cordova-plugin-mfp-push plug in. You don't need this plug in for iOS because the push client-side support for iOS is included in the core mfp plugin.	Yes
Cordova plug-ins management	Yes	Yes	No
Mobile Browser Simulator	Yes **Restriction:** Not all MobileFirst JavaScript APIs are supported by the simulator but you can test calls to adapters with WLResourceRequest. For more information, see "Previewing Cordova web resources with the Mobile Browser Simulator" on page 7-109.	Yes	Yes
Cordova platform management	Yes (Cordova)	Yes (Cordova)	Yes (Proprietary)
MESSAGES (i18n)	Yes	Yes	Yes
Token licensing	Yes	Yes	Yes
Application optimizations			
Minification	Yes (Cordova) **Note:** Use common open source tools. For more information, see "Minification" on page 5-51.	Yes (Cordova) **Note:** Use common open source tools.	Yes (Proprietary)
Concatenation of JS and CSS	Yes (Cordova) **Note:** Use common open source tools.	Yes (Cordova) **Note:** Use common open source tools.	Yes (Proprietary)
Obfuscation	Yes (Cordova) **Note:** Use common open source tools.	Yes (Cordova) **Note:** Use common open source tools.	Yes (Proprietary)

Table 5-18. Cordova developed with IBM MobileFirst Platform Foundation V8.0.0 versus Cordova and hybrid apps developed with IBM MobileFirst Platform Foundation V7.1. (continued)

Feature	**Cordova app with IBM MobileFirst Platform Foundation V8.0.0**	**Cordova app with IBM MobileFirst Platform Foundation V7.1**	**MobileFirst hybrid app with IBM MobileFirst Platform Foundation V7.1**
Android Pro Guard	Yes **Note:** IBM MobileFirst Platform Foundation V8.0.0 does not include the predefined `proguard-project.txt` configuration file for Android ProGuard obfuscation with a MobileFirst Android application.	Yes **Note:** See Android documentation to enable Pro Guard.	Yes

Migrating existing hybrid or cross-platform apps to Cordova apps supported by MobileFirst version 8.0:

You can migrate existing hybrid or cross-platform (Cordova) apps that were developed with IBM MobileFirst Platform Foundation version 6.2 or later to Cordova apps that are supported by IBM MobileFirst Platform Foundation V8.0.0.

Starting the Cordova app migration with the migration assistance tool:

The migration assistance tool helps you prepare your cross-platform apps that were created with earlier versions of IBM MobileFirst Platform Foundation for migraiton by identifying APIs that are no longer valid and copying the projects into Cordova apps that are supported by version 8.0.

Before you begin

The following information is important to know before you use the migration assistance tool:

- You must have an existing IBM MobileFirst Platform Foundation hybrid application or a Cordova application that you created with the **`mfp cordova create`** command.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.
- You must have the Cordova Command-Line Interface (CLI) installed, and any prerequisites installed that are required for using the Cordova CLI for your target platforms. For more information, see The Command-Line Interface at the Apache Cordova website.
- Review and understand the limitations of the migration process. For more information, see "Migrating apps from earlier releases" on page 5-1.

About this task

Cross-platform apps that were created with earlier versions of IBM MobileFirst Platform Foundation commands or the Cordova with IBM MobileFirst Platform Foundation commands are not supported in version 8.0 without some changes. The migration assistance tool simplifies the process with the following functions:

- Scans the JavaScript files in the existing hybrid app or Cordova with IBM MobileFirst Platform Foundation app and identifies APIs that are deprecated, no longer supported, or modified in version 8.0.
- Copies the structure, script, and configuration files of the initial hybrid app or Cordova with IBM MobileFirst Platform Foundation app to a Cordova structure that is supported in version 8.0.

The migration assistance tool does not modify or move any developer code or comments of your app. You must continue the migration process with either "Completing migration of a MobileFirst hybrid app" on page 5-41 or "Completing migration of a MobileFirst Cordova app" on page 5-43 after you run this tool.

Procedure

1. Download the migration assistance tool by using one of the following methods:
   - Download the `.tgz` file from the Jazzhub repository.
   - Download the Developer Kit, which contains the migration assistance tool as a file named `mfpmigrate-cli.tgz`, from the Download page. For more information about the Developer Kit, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.
   - Download the tool by using the instructions that are provided in "Opening the MobileFirst Operations Console" on page 7-12.
2. Install the migration assistance tool.
   a. Change to the directory where you downloaded the `.tgz` file.
   b. Use NPM to install the tool by entering the following command:
      ```
      npm install -g tgz_filename
      ```
3. Scan and copy the IBM MobileFirst Platform Foundation app by entering the following command:
   ```
   mfpmigrate client --in source_directory --out destination_directory
   ```
   source_directory
   The current location of the initial project.

   destination_directory
   The directory where the new version 8.0 compatible Cordova structure is output.

   When it is used with the client command, the migration assistance tool completes the following actions:
   - Identifies APIs in the existing IBM MobileFirst Platform Foundation app that are removed, deprecated, or changed in version 8.0.
   - Creates a Cordova structure based on the structure of the initial app.
   - Copies or adds the following items, when applicable:
     - Android operating system
     - iPhone and iPad operating system
     - Windows operating system
     - Cordova-mfp-plugin
     - Cordova-plugin-mfp-jsonstore plug-in, if the JSONStore feature was installed on the old project.
     - Cordova-plugin-mfp-fips plug-in, if the FIPS feature was installed on the old project.
     - Cordova-plugin-mfp-push plug-in, if the push notification feature was installed on the old project.

- Hybrid certificates, if certificate pinning was enabled on the old project.
- Application, script, and XML files

Important: The migration assistance tool does not copy developer code or commented text into the new structure.

4. Resolve the API issues in the new Cordova app.
   a. Review the `api-report.html` file that is created in the `destination_directory` directory. Each row of the table in this file identifies a deprecated, changed, or removed API that is used in the app that is not compatible with version 8.0. This file also specifies the replacement for removed APIs, when one is available.

Table 5-19. Example of a table in the `api-report.html` file

File path	**Line number**	**API**	**Line content**	**Category of API change**	**Description of change with instructions about how to resolve the API change.**
c:\local\ Cordova\ www\js\ index.js	15	Wl.Client.getAppProperty	document.getElementById ('app_version') textContent= WL.Client.getAppProperty ("APP_VERSION");	Not supported	Removed from 8.0. Use Cordova plug-in to get app version. No replacement API. More information.

   b. Address the API issues that are identified in the `api-report.html` file.
5. Manually copy the developer code from the initial app structure into the correct location in the new Cordova structure. Copy the content in the following directories, according to the type of the source IBM MobileFirst Platform Foundation app:

   IBM MobileFirst Platform Foundation hybrid app
   Copy the contents of the `common` directory of the source app to the `www` directory in your new Cordova app.

   Cordova with IBM MobileFirst Platform Foundation app
   Copy the contents of the `www` directory of the source app to the `www` directory in your new Cordova app.
6. Run the migration assistance tool with the scan command on your new app to confirm that your API changes are complete.
   a. Enter the following command to run the scan:

```
mfpmigrate scan --in source_directory
--out destination_directory --type hybrid
```

   source_directory
   The current location of the files to scan. In an IBM MobileFirst Platform Foundation hybrid app, this location is the `common` directory of your app. In an IBM MobileFirst Platform Foundation version 8.0 Cordova cross-platform app, this location is the `www` directory.

   destination_directory
   The directory where your scan results are output.

scan_type
: The type of project to scan.

   b. Address any remaining API issues that are identified in the `api-report.html` file.

7. Repeat step 6 on page 5-40 to run the scan tool on the new Cordova app until all of the issues are resolved.

What to do next

Complete the migration by completing the steps in one of the following topics, depending on the type of app that you are migrating:

- If you are migrating an IBM MobileFirst Platform Foundation hybrid app, continue with "Completing migration of a MobileFirst hybrid app."
- If you are migrating a Cordova app with IBM MobileFirst Platform Foundation, continue with "Completing migration of a MobileFirst Cordova app" on page 5-43.

Completing migration of a MobileFirst hybrid app:

After you use the migration assistance tool, you must modify some portions of your code manually to complete the migration process.

Before you begin

- You must have already run the **`mfpmigrate`** migration assistance tool on your existing hybrid app. For more information, see "Starting the Cordova app migration with the migration assistance tool" on page 5-38.
- You must have the Cordova Command-Line Interface (CLI) installed, and any prerequisites installed that are required for using the Cordova CLI for your target platforms if you need to install any additional Cordova plug-ins. (See step 6.) For more information, see The Command-Line Interface at the Apache Cordova web site.
- You must have internet access if you need to download a new version of JQuery (step 1c) or if you need to install any additional Cordova plug-ins (step 6).
- You must have node.js version 4.0.0 or later installed if you need to install additional Cordova plug-ins (step 6).

About this task

Complete the steps in this task to finish migrating your MobileFirst hybrid application from IBM MobileFirst Platform Foundation 7.1 to a Cordova application that includes support for IBM MobileFirst Platform Foundation 8.0.

After you complete the migration, your app can use Cordova platforms and plug-ins that you obtain independently of IBM MobileFirst Platform Foundation, and you can continue to develop the app with your preferred Cordova development tools.

Procedure

1. Update the `www/index.html` file.
   a. Add the following CSS code to the head of your `index.html` file, before your CSS code that is already there.
   ```
   <link rel="stylesheet" href="worklight/worklight.css">
   <link rel="stylesheet" href="css/main.css">
   ```

Note: The `worklight.css` file sets the body attribute to `relative`. If this affects the style of your app, then declare a different value for the position in your own CSS code. For example:

```
body {
  position: absolute;
}
```

b. Add Cordova JavaScript to the head of the file after the CSS definitions.

```
<script type="text/javascript" src="cordova.js"></script>
```

c. Remove the following line of code if it is present.

```
<script>window.$ = window.jQuery = WLJQ;</script>
```

You can download your own version of JQuery, and load it as shown in the following code line.

```
<script src="lib/jquery.min.js"></script>
```

You do not have to move the optional jQuery addition to the `lib` folder. You can move this addition anywhere you want to, but you must correctly reference it in the `index.html` file.

2. Update the `www/js/InitOptions.js` file to call `WL.Client.init` automatically.
   a. Remove the following code from `InitOptions.js`.

      The function `WL.Client.init` is called automatically with the global variable `wlInitOptions`.

```
if (window.addEventListener) {
  window.addEventListener('load', function() { WL.Client.init(wlInitOptions); }, false);
} else if (window.attachEvent) {
  window.attachEvent('onload',  function() { WL.Client.init(wlInitOptions); });
}
```

3. Optional: Update the `www/InitOptions.js` to call WLClient.init manually.
   a. Edit the `config.xml` file and set the `<mfp:clientCustomInit>` element's `enabled` attribute to `true`.
   b. If you are using the MobileFirst hybrid default template, replace this code:

```
if (window.addEventListener) {
  window.addEventListener('load', function() { WL.Client.init(wlInitOptions); }, false);
} else if (window.attachEvent) {
  window.attachEvent('onload',  function() { WL.Client.init(wlInitOptions); });
}
```

with the following code:

```
if (document.addEventListener) {
  document.addEventListener('mfpready', function() { WL.Client.init(wlInitOptions); }, false);
} else if (window.attachEvent) {
  document.attachEvent('mfpready',  function() { WL.Client.init(wlInitOptions); });
}
```

4. Optional: If you have logic specific to a hybrid environment, for example in *Your app*`/iphone/js/main.js`, copy the function `wlEnvInit()` and append it at the end of `www/main.js`.

```
// This wlEnvInit method is invoked automatically by MobileFirst runtime after successful initialization.
function wlEnvInit() {
  wlCommonInit();
  if (cordova.platformId === "ios") {
    // Environment initialization code goes here for ios
  } else if (cordova.platformId === "android") {
    // Environment initialization code goes here for android
  }
}
```

5. Optional: If your original application uses the FIPS feature, change the JQuery event listener to a JavaScript event listener that listens to the `WL/FIPS/READY` event. For more information about FIPS, see "FIPS 140-2 support" on page 10-75.
6. Optional: If your original application uses any third-party Cordova plug-ins that are not replaced or supplied by the migration assistance tool, manually add the plug-ins to the Cordova app with the **`cordova plugin add`** command. For information about which plug-ins are replaced by the tool, see "Starting the Cordova app migration with the migration assistance tool" on page 5-38.

Results

You now have a Cordova app that you can continue to develop with your preferred Cordova tools, but that also includes MobileFirst functionality.

What to do next

Register your app to a MobileFirst Server. For more information, see "Registering Cordova applications from the MobileFirst Platform CLI" on page 7-107.

Completing migration of a MobileFirst Cordova app:

After you use the migration assistance tool, you must modify some portions of your code manually to complete the migration process.

Before you begin

- You must have already run the **`mfpmigrate`** migration assistance tool on your existing Cordova app. For more information, see "Starting the Cordova app migration with the migration assistance tool" on page 5-38.
- You must have the Cordova Command-Line Interface (CLI) installed, and any prerequisites installed that are required for using the Cordova CLI for your target platforms. For more information, see The Command-Line Interface at the Apache Cordova web site.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.

About this task

The Cordova app that you created with **`mfp cordova create`** uses the Cordova platform and plug-in versions that were supplied with IBM MobileFirst Platform Foundation previous version. After you complete the migration, your migrated app can use Cordova platforms and plug-ins that you obtain independently of IBM MobileFirst Platform Foundation. This is the only type of support for Cordova applications that is available with IBM MobileFirst Platform Foundation V8.0.0.

To migrate, you run the migration assistance tool and then make other modifications to your app.

Procedure

1. With the Cordova development tool of your choice, add any Cordova plug-ins other than Cordova plug-ins that enable MobileFirst features that were in your original application. For example, with the Cordova CLI, to add the plug-ins `cordova-plugin-file` and `cordova-plugin-file-transfer`, enter:

   ```
   cordova plugin add cordova-plugin-file cordova-plugin-file-transfer
   ```

Note: The **`mfpmigrate`** migration assistance tool added the Cordova plug-ins for MobileFirst features, so you do not have to add them. For more information about these plug-ins, see "Cordova plug-ins for MobileFirst features" on page 7-87.

2. Optional: If your original application uses the FIPS feature, change the JQuery event listener to a JavaScript event listener that listens to the `WL/FIPS/READY` event. For more information about FIPS, see "FIPS 140-2 support" on page 10-75.
3. Optional: If your original application uses any third-party Cordova plug-ins that are not replaced or supplied by the migration assistance tool, manually add the plug-ins to the Cordova app with the **`cordova plugin add`** command. For information about which plug-ins are replaced by the tool, see "Starting the Cordova app migration with the migration assistance tool" on page 5-38.
4. Optional: (Only for apps that include the iOS platform, and that use OpenSSL.) Add the `cordova-plugin-mfp-encrypt-utils` plug-in to your app. The `cordova-plugin-mfp-encrypt-utils` plug-in provides iOS OpenSSL frameworks for encryption for Cordova applications with the iOS platform. For more information, see "Enabling OpenSSL for Cordova iOS" on page 7-126 and "Adding MobileFirst features to an existing Cordova app" on page 7-91.

Results

You now have a Cordova app that you can continue to develop with your preferred Cordova tools, but that also includes MobileFirst functionality.

What to do next

Register your app to a MobileFirst Server. For more information, see "Registering Cordova applications from the MobileFirst Platform CLI" on page 7-107.

Migrating encryption for iOS Cordova:

If your iOS Hybrid or Cordova application used OpenSSL encryption, you may want to migrate your app to the new V8.0.0 native encryption. If you want to continue using OpenSSL you need to add an additional Cordova plug-in.

For more information on the iOS Cordova encryption options for migration see "Migration options" on page 7-126 section within the "Enabling OpenSSL for Cordova iOS" on page 7-126 topic.

Migrating Direct Update:

Learn how to migrate Direct Update for V8.0.0. Direct Update is triggered after the first access to a protected resource. The process to deploy new web resources has changed in V8.0.0.

Unlike in previous versions, in V8.0.0, if an application does not access a secure MobileFirst resource, the client application does not receive updates, even if updates are available on the server. A resource might be unprotected, for example because OAuth has been disabled by the annotation **`@OAuth(security=false)`** or by configuration. You can work around this risk in one of the following ways:

- Explicitly obtain an access token. See the obtainAccessToken API in the class.
- Call another protected resource. See the class.

To use Direct Update: Starting with V8.0.0, you no longer upload a `.wlapp` file to MobileFirst Server. Instead, you upload a smaller web resource archive (.zip file). The archive file no longer contains the web preview files or skins that were widely used in previous versions. These have been discontinued. The archive contains only the web resources that are sent to the clients, as well as checksums for Direct Update validations. For more information, see "Updating Cordova client apps directly" on page 7-235.

Upgrading the WebView:

IBM MobileFirst Platform Foundation V8.0.0 Cordova SDK (JavaScript) introduced numerous changes that require adaptations of your code.

The manual migration process involves a few stages:
- Creating a new Cordova MobileFirst project
- Replacing the necessary web resource elements with the code from your previous version
- Making the necessary changes to your JavaScript code to conform to SDK changes

Many MobileFirst API elements were removed in V8.0.0. Removed elements are clearly marked as non-existent in an IDE that supports autocorrect for JavaScript.

For more information about IDEs for editing JavaScript with autocomplete, see "Editing WebView (JavaScript) code" on page 7-129.

Table 1 lists those API elements that require removal, with suggestions on how to replace the functionality. Many of the removed elements are UI elements that can be replaced with Cordova plug-ins or HTML 5 elements. Some methods have changed.

For examples of some sample code for creating the client and accessing the server with the new V8.0.0 client for JavaScript, see "Some initial WebView code for connecting to the server" on page 7-130.

Table 5-20. Discontinued JavaScript UI elements

API element	**Migration path**

`WL.BusyIndicator`
`WL.OptionsMenu`
`WL.TabBar`
`WL.TabBarItem`
	Use Cordova plug-ins or HTML 5 elements.

`WL.App.close()`	Handle this event outside of MobileFirst.
`WL.App.copyToClipboard()`	Use Cordova plug-ins providing this functionality.
`WL.App.openUrl(url, target, options)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova `InAppBrowser` plug-in provides this feature..

Table 5-20. Discontinued JavaScript UI elements (continued)

API element	Migration path

`WL.App.overrideBackButton(callback)`
 	
`WL.App.resetBackButton()`
	Use Cordova plug-ins providing this functionality.
	Note: For your information, the Cordova `backbutton` plug-in provides this feature..

`WL.App.getDeviceLanguage()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova cordova-plugin-globalization plug-in provides this feature.
`WL.App.getDeviceLocale()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova cordova-plugin-globalization plug-in provides this feature.
`WL.App.BackgroundHandler`	To run a custom handler function, use the standard Cordova pause event listener. Use a Cordova plug-in that provides privacy and prevents iOS and Android systems and users from taking snapshots or screen captures. For more information, see the description of the PrivacyScreenPlugin at https://github.com/devgeeks/PrivacyScreenPlugin.

`WL.Client.close()`
 	
`WL.Client.restore()`
 	
`WL.Client.minimize()`
	The functions were provided to support the Adobe AIR platform, which is not supported by IBM MobileFirst Platform V8.0.0.

`WL.Toast.show(string)`	Use Cordova plug-ins for Toast.

Table 5-21. Other Discontinued JavaScript elements

API	Migration path
`WL.Client.checkForDirectUpdate(options)`	No replacement. **Note:** You can call WLAuthorizationManager.obtainAccessToken to trigger a direct update if one is available. The access to a security token triggers a direct update if one is available on the server. But you cannot trigger Direct Update on demand. For more information about customizing the Direct Update user interface and process, see "Customizing the Direct Update user interface and process" on page 7-244.

`WL.Client.setSharedToken({key: myName, value: myValue})`
 	
`WL.Client.getSharedToken({key: myName})`
 	
`WL.Client.clearSharedToken({key: myName})`
	No replacement.

`WL.Client.isConnected()`
 	
`connectOnStartup init option`
	Use WLAuthorizationManager.obtainAccessToken to check connectivity to the server and apply application management rules.

Table 5-21. Other Discontinued JavaScript elements (continued)

API	Migration path

`WL.Client.setUserPref(key,value, options)`
 	
`WL.Client.setUserPrefs(userPrefsHash, options)`
 	
`WL.Client.deleteUserPrefs(key, options)`
	No replacement. You can use an adapter and the MFP.Server.getAuthenticatedUser API to manage user preferences.

`WL.Client.getUserInfo(realm, key)`
 	
`WL.Client.updateUserInfo(options)`
	No replacement.

`WL.Client.logActivity(activityType)`	Use WL.Logger. For more information, see "Logger SDK" on page 11-37.
`WL.Client.login(realm, options)`	Use WLAuthorizationManager.login. To get started with authentication and security, see the Authentication and Security tutorials.
`WL.Client.logout(realm, options)`	Use WLAuthorizationManager.logout.
`WL.Client.obtainAccessToken(scope, onSuccess, onFailure)`	Use WLAuthorizationManager.obtainAccessToken.

`WL.Client.transmitEvent(event, immediate)`
 	
`Wl.Client.purgeEventTransmissionBuffer()`
 	
`Wl.Client.setEventTransmissionPolicy(policy)`
	Create a custom adapter for receiving notifications of these events.

`WL.Device.getContext()`
 	
`WL.Device.startAcquisition(policy, triggers, onFailure)`
 	
`WL.Device.stopAcquisition()`
 	
`WL.Device.Wifi`
 	
`WL.Device.Geo.Profiles`
 	
`WL.Geo`
	Use native API or third-party Cordova plug-ins for GeoLocation.

`WL.Client.makeRequest (url, options)`	Create a custom adapter that provides the same functionality
`WLDevice.getID(options)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, device.uuid from the cordova-plugin-device plug-in provides this feature.
`WL.Device.getFriendlyName()`	Use WL.Client.getDeviceDisplayName
`WL.Device.setFriendlyName()`	Use WL.Client.setDeviceDisplayName
`WL.Device.getNetworkInfo(callback)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the cordova-plugin-network-information plug-in provides this feature.
`WLUtils.wlCheckReachability()`	Create a custom adapter to check server availability.
`WL.EncryptedCache`	Use JSONStore to store encrypted data locally. JSONStore is in the cordova-plugin-mfp-jsonstore plug-in. For more information, see "JSONStore" on page 7-134.
`WL.SecurityUtils.remoteRandomString(bytes)`	Create a custom adapter that provides the same functionality.

Table 5-21. Other Discontinued JavaScript elements (continued)

API	Migration path
`WL.Client.getAppProperty(property)`	You can retrieve the app version property by using the `cordova plugin add cordova-plugin-appversion` plug-in. The version that is returned is the native app version (Android and iOS only).
`WL.Client.Push.*`	Use "JavaScript client-side push API" on page 8-4 from the cordova-plugin-mfp-push plug-in. For more information, see "Migrating to push notifications from event source-based notifications" on page 5-54.
`WL.Client.Push.subscribeSMS(alias, adapterName, eventSource, phoneNumber, options)`	Use MFPPush.registerDevice(org.json.JSONObject options, MFPPushResponseListener listener) to register the device for push and SMS.
`WLAuthorizationManager.obtainAuthorizationHeader(scope)`	Use WLAuthorizationManager.obtainAccessToken to obtain a token for the required scope. For more information about implementing a custom resource request, see "JavaScript custom resource-request implementation sample" on page 7-311.
`WLClient.getLastAccessToken(scope)`	Use WLAuthorizationManager.obtainAccessToken

`WLClient.getLoginName()`
 	
`WL.Client.getUserName(realm)`
	No replacement

`WL.Client.getRequiredAccessTokenScope(status, header)`	Use WLAuthorizationManager.isAuthorizationRequired and WLAuthorizationManager.getResourceScope.
`WL.Client.isUserAuthenticated(realm)`	No replacement
`WLUserAuth.deleteCertificate(provisioningEntity)`	No replacement
`WL.Trusteer.getRiskAssessment(onSuccess, onFailure)`	No replacement
WL.Client.createChallengeHandler(realmName)	To create a challenge handler for handling custom gateway challenges, use WL.Client.createGatewayChallengeHandler(gatewayName). To create a challenge handler for handling MobileFirst security-check challenges, use WL.Client.createSecurityCheckChallengeHandler(securityCheckName). For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
WL.Client.createWLChallengeHandler(realmName)	Use WL.Client.createSecurityCheckChallengeHandler(securityCheckName). For more information about the V8.0.0 challenge-handler APIs, see "Client security APIs" on page 7-305.
challengeHandler.isCustomResponse() where *challengeHandler* is a challenge-handler object that is returned by WL.Client.createChallengeHandler()	Use gatewayChallengeHandler.canHandleResponse() where *gatewayChallengeHandler* is a challenge-handler object that is returned by WL.Client.createGatewayChallengeHandler().
wlChallengeHandler.processSucccess() where *wlChallengeHandler* is a challenge-handler object that is returned by WL.Client.createWLChallengeHandler()	Use securityCheckChallengeHandler.handleSuccess() where *securityCheckChallengeHandler* is a challenge-handler object that is returned by WL.Client.createSecurityCheckChallengeHandler().

Table 5-21. Other Discontinued JavaScript elements (continued)

API	Migration path
WL.Client.AbstractChallengeHandler.submitAdapterAuthentication()	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use a challenge-handler object that is returned by WL.Client.createGatewayChallengeHandler. For MobileFirst security-check challenge handlers, use a challenge-handler object that is returned by WL.Client.createSecurityCheckChallengeHandler.
WL.Client.AbstractChallengeHandler.submitFailure(err)	Use WL.Client.AbstractChallengeHandler.cancel()WL.Client.AbstractChal
WL.Client.createProvisioningChallengeHandler()	No replacement. Device provisioning is now handled automatically by the security framework.

Table 5-22. Deprecated JavaScript APIs

API	Migration path

`WLClient.invokeProcedure(WLProcedureInvocationData invocationData,WLResponseListener responseListener)`
 	
 `WL.Client.invokeProcedure(invocationData, options)`
 	
 `WLClient.invokeProcedure(WLProcedureInvocationData invocationData, WLResponseListener responseListener, WLRequestOptions requestOptions)`
 	
 `WLProcedureInvocationResult`
	Use the WLResourceRequest instead.
	 Note: The implementation of invokeProcedure uses WLResourceRequest.

`WLClient.getEnvironment`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the device.platform plug-in provides this feature.
`WLClient.getLanguage`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the cordova-plugin-globalization plug-in provides this feature.
`WL.Client.connect(options)`	Use WLAuthorizationManager.obtainAccessToken to check connectivity to the server and apply application management rules.

Removed components:

The Cordova project created by MobileFirst 7.1 Studio included many resources that supported propriety functionality. However in V8.0.0 only pure Cordova is supported and the MobileFirst API no longer supports these features.

Skins

MobileFirst application **skins** provided a way of optimizing the UI for adapting to different devices and formats and is no longer supported in V8.0.0.

To replace this type of functionality it is recommended to adopt responsive web design methods provided by Cordova and HTML 5.

Shells

Shells allowed the development of a set of functionalities to be used by and shared among applications. In this way developers who were more experienced with the native environment could provide a set of core functions. These shells were bundled into **inner applications** and used by developers who are involved with business logic or UI development.

If the previous hybrid app used shells and inner applications, it is recommended to adopt Cordova design patterns and implement the shell components as Cordova plug-ins, that can be shared across applications. Developers may find ways to reuse parts of shell code and migrate them to Cordova plug-ins.

For example, if a customer has a set of web resources (JavaScript, css files, graphics, html) that are common across all their apps they can create a Cordova plug-in that copies these resources into the app's `www` folder.

Let's say these resources are within the `src/www/acme/` folder:

`src/www/acme/js/acme.js`

`src/www/acme/css/acme.css`

`src/www/acme/img/acme-logo.png`

`src/www/acme/html/banner.html`

`src/www/acme/html/footer.html`

`plugin.xml`

The `plugin.xml` file contains the `<asset>` tag, containing the `source` and `target` for copying the resources:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
     xmlns="http://apache.org/cordova/ns/plugins/1.0"
     xmlns:rim="http://www.blackberry.com/ns/widgets"
     xmlns:android="http://schemas.android.com/apk/res/android"
     id="cordova-plugin-acme"
     version="1.0.1">
<name>ACME Company Shell Component</name>
<description>ACME Company Shell Component</description>
<license>MIT</license>
<keywords>cordova,acme,shell,components</keywords>
<issue>https://www.acme.com/support</issue>
<asset src="src/www/acme" target="www/acme"/>
</plugin>
```

After the `plugin.xml` is added to the Cordova `config.xml` file, the resources listed in the `asset src` are copied to the `asset target` during compilation.

Then in their `index.html` file or anywhere inside their app they can reuse these resources.

```
<link rel="stylesheet" type="text/css" href="acme/css/acme.css">
<script type="text/javascript" src="acme/js/acme.js"></script>
<div id="banner"></div>
<div id="app"></div>
<div id="footer"></div>
```

```
<script type="text/javascript">
    $("#banner").load("acme/html/banner.html");
    $("#footer").load("acme/html/footer.html");
</script>
```

Settings page

The **settings page** was a UI available in the MobileFirst hybrid app that allowed the developer to change the server URL at runtime for testing purposes. The developer can now use existing MobileFirst Client API to change the server URL at runtime. For more information, see WL.App.setServerUrl.

Minification

MobileFirst Studio 7.1 provided an OOTB method of reducing the size of your JavaScript code by removing all unnecessary characters before compilation. This removed functionality can be replaced by adding Cordova hooks to your project.

Many hooks are available for minifying your `Javascript` and `css` files and can be placed in the `config.xml` at the `before_prepare` event.

Here are some recommended hooks:
- https://www.npmjs.com/package/uglify-js
- https://www.npmjs.com/package/clean-css

These hooks can be defined in either a plug-in file or in the app's `config.xml` file, using the `<hook>` elements.

In this example, using the `before_prepare` hook event, a script is run for minifying before `cordova prepare` copies the files to each platform's `www/` folder:

```
<hook type="before_prepare" src="scripts/uglify.js" />
```

Migrating existing adapters to work under MobileFirst Server V8.0.0

Starting with V8.0.0 of MobileFirst Server, adapters are Maven projects. Learn how to upgrade adapters that were developed under earlier versions of MobileFirst Server.

Before you begin

This page describes the steps to take to migrate adapters that were developed to work with MobileFirst Server V6.2 or later so that they work with MobileFirst Server V8.0.0.

To start, study the changes in adapter APIs that are described in "Deprecated features and API elements" on page 3-17 and "Server-side API changes in V8.0.0" on page 5-14.
- Under certain conditions, existing adapters work as-is with MobileFirst Server V8.0.0. See "Using older adapters as-is under MobileFirst Server V8.0.0" on page 5-52.
- In most cases, you need to upgrade the adapters. For Java adapters, see "Migrating Java adapters to Maven projects for MobileFirst Server V8.0.0" on page 5-52. For JavaScript adapters, see "Migrating JavaScript adapters to Maven projects for MobileFirst Server V8.0.0" on page 5-54.

Using older adapters as-is under MobileFirst Server V8.0.0
About this task

An existing adapter can be deployed as-is under MobileFirst Server V8.0.0, unless it matches any of the following criteria:

Table 5-23. Adapter conditions.

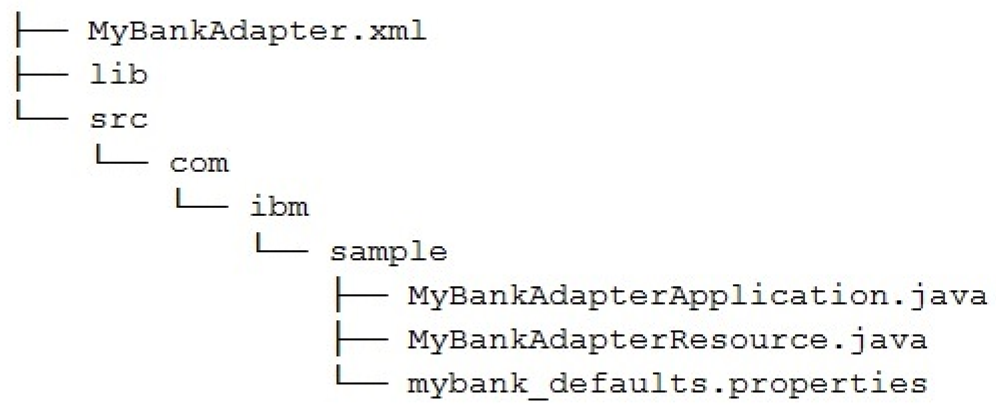
Adapter type	Condition
Java	Uses the PushAPI or SecurityAPI interfaces
JavaScript	Was built using IBM Worklight V6.2 or earlier
	Uses a connection type that is not HTTP or SQL
	Contains procedures with securityTest customization
	Contains procedures that use the user identity to connect to the back end
	Uses any of the following APIs: • WL.Device.* • WL.Geo.* • WL.Server.readSingleJMSMessage • WL.Server.readAllJMSMessages • WL.Server.writeJMSMessage • WL.Server.requestReplyJMSMessage • WL.Server.getActiveUser • WL.Server.setActiveUser • WL.Server.getCurrentUserIdentity • WL.Server.getCurrentDeviceIdentity • WL.Server.createEventSource • WL.Server.createDefaultNotification • WL.Server.getUserNotificationSubscription • WL.Server.notifyAllDevices • WL.Server.notifyDeviceToken • WL.Server.notifyDeviceSubscription • WL.Server.sendMessage • WL.Server.createEventHandler • WL.Server.setEventHandlers • WL.Server.setApplicationContext • WL.Server.fetchNWBusinessObject • WL.Server.createNWBusinessObject • WL.Server.deleteNWBusinessObject • WL.Server.updateNWBusinessObject • WL.Server.getBeaconsAndTriggers • WL.Server.signSoapMessage • WL.Server.createSQLStatement

Migrating Java adapters to Maven projects for MobileFirst Server V8.0.0
Procedure

1. Create a Maven adapter project with the archetype **`adapter-maven-archetype-java`**. When setting the parameter **`artifactId`** use the adapter name and for the

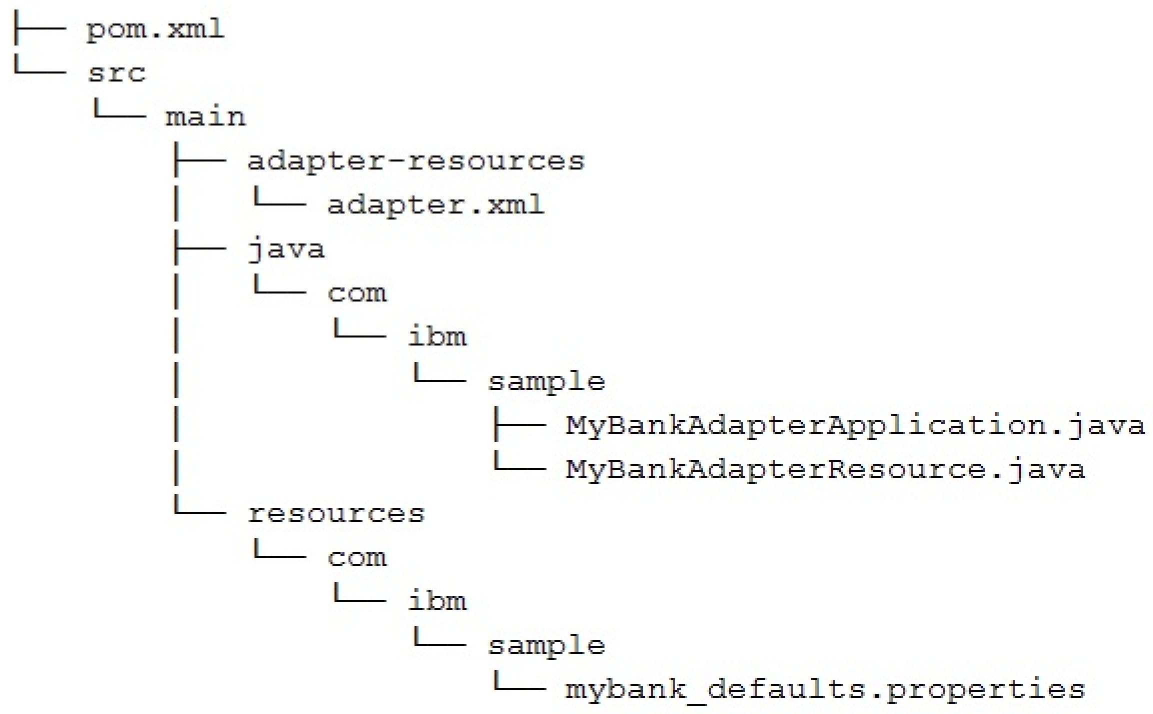
parameter **package** use the same package as the one in the existing Java adapter. For more information, see “Creating adapters with Maven” on page 7-196.

2. Overwrite the adapter-descriptor file (`adapter.xml`) under `src/main/adapter-resources` in the created project from the existing Java adapter. For more details about the descriptor, see “The Java adapter-descriptor file” on page 7-194.
3. Remove all the files under `src/main/java` in the created project from the existing Java adapter, then copy all the Java files under the old adapter's `src` folder, but preserve the same folder structure. Copy all the non-Java files under the `src` folder of the old adapter to the `src/main/resources` of the new adapter. By default, `src/main/resources` does not exist, so if the adapter contains non-Java files, create it. For the changes in Java adapter APIs, see “Server-side API changes in V8.0.0” on page 5-14. The following diagrams illustrate the structure of adapters up to V7.1 and Maven adapters, starting from V8.0:



```
├── MyBankAdapter.xml
├── lib
└── src
    └── com
        └── ibm
            └── sample
                ├── MyBankAdapterApplication.java
                ├── MyBankAdapterResource.java
                └── mybank_defaults.properties
```

Figure 5-2. Adapter folder structure up to V7.1



```
├── pom.xml
└── src
    └── main
        ├── adapter-resources
        │   └── adapter.xml
        ├── java
        │   └── com
        │       └── ibm
        │           └── sample
        │               ├── MyBankAdapterApplication.java
        │               └── MyBankAdapterResource.java
        └── resources
            └── com
                └── ibm
                    └── sample
                        └── mybank_defaults.properties
```

Figure 5-3. Maven adapter structure, starting from V8.0.0

4. Using either of the following methods, add any JAR files that are not in the Maven repository:
   - Add the JAR files to a local repository, as described in Guide to installing third-party JARs, then add them to dependencies element.
   - Add the JAR files to the dependencies element by using the systemPath element. For more information, see Introduction to the Dependency Mechanism.

Migrating JavaScript adapters to Maven projects for MobileFirst Server V8.0.0

Procedure

1. Create a Maven adapter project with the archetype **adapter-maven-archetype-http** or **adapter-maven-archetype-sql**. When setting the parameter **artifactId** use the adapter name. For more information, see "Creating adapters by using Maven" on page 7-214.
2. Overwrite the adapter-descriptor file (`adapter.xml`) under `src/main/adapter-resources` in the created project from the existing JavaScript adapter. For details about the descriptor, see "The JavaScript adapter-descriptor file" on page 7-206.
3. Overwrite the JavaScript files `src/main/adapter-resources/js` in the created project from the existing JavaScript adapter JavaScript files.

Migrating to push notifications from event source-based notifications

From IBM MobileFirst Platform Foundation V8.0.0, the event source-based model is not supported, and push notifications capability is enabled entirely by the push service model. For existing event source-based applications on earlier versions of MobileFirst to be moved to V8.0.0, they must be migrated to the new push service model.

About this task

During migration, keep in mind that it is not about using one API instead of another, but more about using one model/approach versus another.

For example, in the event source-based model, if you were to segment your mobile application users to send notifications to specific segments, you would model every segment as a distinct event source. In the push service model, you would achieve the same by defining tags that represents segments and have users subscribe to the respective tags. Tag-based notifications is a replacement to event source-based notifications.

Table 5-24 provides you with a comparison between the two models.

Table 5-24. Event source-based model versus push service model

User requirement	Event source model	Push service model
To enable your application with push notifications	• Create an Event Source Adapter and within it create an EventSource. • Configure or setup your application with push credentials.	• Configure or setup your application with push credentials.
To enable your mobile client application with push notifications	• Create WLClient • Connect to the MobileFirst Server • Get an instance of push client • Subscribe to the Event source	• Instantiate push client • Initialize push client • Register the mobile device
To enable your mobile client application for notifications based on specific tags	Not supported.	Subscribe to the tag (that uses tag name) that is of interest.
To receive and handle notifications in your mobile client applications	Register a listener implementation.	Register a listener implementation.

Table 5-24. Event source-based model versus push service model (continued)

User requirement	Event source model	Push service model
To send push notifications to mobile client applications	• Implement adapter procedures that internally call the WL.Server APIs to send push notifications. • WL Server APIs provide means to send notifications: – By user – By device – Broadcasts (all devices) • Backend server applications can then invoke the adapter procedures to trigger push notification as part of their application logic.	• Backend server applications can directly call the messages REST API. However, these applications must register as confidential client with the MobileFirst Server and obtain a valid OAuth access token that must be passed in the Authorization header of the REST API. • The REST API provides options to send notifications: – By user – By device – By platform – By tags – Broadcasts (all devices)
To trigger push notifications as regular time periods (polling intervals)	Implement the function to send push notifications within the event-source adapter and this as part of the createEventSource function call.	Not supported.
To register a hook with the name, URL, and the even types.	Implement hooks on the path of a device subscribing or unsubscribing to push notifications.	Not supported.

Migration scenarios

Starting from IBM MobileFirst Platform FoundationV8.0.0, the event source-based model will not be supported and push notifications capability will be enabled on IBM MobileFirst Platform Foundation entirely by the push service model, which is a more simple and agile alternative to event source model.

Existing event source-based applications on earlier versions of IBM MobileFirst Platform Foundation need to be migrated to V8.0.0, to the new push service model.

For more information, see "Migrating to push notifications from event source-based notifications" on page 5-54

Hybrid applications:

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application:

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in V8.0.0, convert this model to Unicast notification.

1. Initialize the MobileFirst push client instance in your application and in the success callback register the callback method that should receive the notification.
   ```
   MFPPush.initialize(function(successResponse){
   MFPPush.registerNotificationsCallback(notificationReceived); },
   function(failureResponse){alert("Failed to initialize");
                             }
   );
   ```
2. Implement the notification callback method.
   ```
   var notificationReceived = function(message) {
         alert(JSON.stringify(message));
   };
   ```
3. Register the mobile device with the push notification service.
   ```
   MFPPush.registerDevice(function(successResponse) {
        alert("Successfully registered");
           },
         function(failureResponse) {
        alert("Failed to register");
           }
       );
   ```
4. (Optional) Un-register the mobile device from the push notification service.
   ```
    MFPPush.unregisterDevice(function(successResponse) {
        alert("Successfully unregistered");
           },
         function(failureResponse) {
        alert("Failed to unregister");
           }
       );
   ```
5. Remove WL.Client.Push.isPushSupported() (if used) and use.
   ```
   MFPPush.isPushSupported (function(successResponse) {
        alert(successResponse);
           },
         function(failureResponse) {
        alert("Failed to get the push suport status");
           }
       );
   ```
6. Remove the following WL.Client.Push APIs, since there will be no event source to subscribe to and register notification callbacks.
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. onReadyToSubscribe()

Server

1. Remove the following WL.Server APIs (if used), in your adapter:
   - notifyAllDevices()
   - notifyDevice()
   - notifyDeviceSubscription()
   - createEventSource()

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 2: Existing applications using multiple event sources in their application:

Applications using multiple event sources requires segmentation of users based on subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this, this model can be converted to tag-based notification.

1. Initialize the MFPPush client instance in your application and in the success callback register the callback method that should receive the notification.
```
MFPPush.initialize(function(successResponse){
     MFPPush.registerNotificationsCallback(notificationReceived);                },
     function(failureResponse){
      alert("Failed to initialize");
     }
  );
```
2. Implement the notification callback method.
```
var notificationReceived = function(message) {
      alert(JSON.stringify(message));
 };
```
3. Register the mobile device with the push notification service.
```
MFPPush.registerDevice(function(successResponse) {
     alert("Successfully registered");
        },
      function(failureResponse) {
     alert("Failed to register");
        }
    );
```
4. (Optional) Unregister the mobile device from the push notification service.
```
MFPPush.unregisterDevice(function(successResponse) {
     alert("Successfully unregistered");
        },
      function(failureResponse) {
     alert("Failed to unregister");
        }
    );
```
5. Remove WL.Client.Push.isPushSupported() (if used) and use.
```
MFPPush.isPushSupported (function(successResponse) {
     alert(successResponse);
        },
      function(failureResponse) {
     alert("Failed to get the push suport status");
        }
   );
```

6. Remove the following WL.Client.Push APIs since there will be no event source to subscribe to and register notification callbacks.
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. onReadyToSubscribe()
7. Subscribe to tags.

```
var tags = ['sample-tag1','sample-tag2']
       MFPPush.subscribe(tags, function(successResponse) {
     alert("Successfully subscribed");
        },
      function(failureResponse) {
     alert("Failed to subscribe");
        }
   );
```

8. (Optional) Unsubscribe from tags.

```
MFPPush.unsubscribe(tags, function(successResponse) {
     alert("Successfully unsubscribed");
        },
      function(failureResponse) {
     alert("Failed to unsubscribe");
        }
   );
```

Server

Remove the following WL.Server APIs (if used) in your adapter:

- **`notifyAllDevices()`**
- **`notifyDevice()`**
- **`notifyDeviceSubscription()`**
- **`createEventSource()`**

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable Push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 3: Existing applications using broadcast/Unicast notification in their application:

Client

1. Initialize the MFPPush client instance in your application and in the success callback register the callback method that should receive the notification.

   ```
   MFPPush.initialize(function(successResponse){
        MFPPush.registerNotificationsCallback(notificationReceived);                   },
        function(failureResponse){
         alert("Failed to initialize");
        }
     );
   ```

2. Implement the notification callback method.

   ```
   var notificationReceived = function(message) {
         alert(JSON.stringify(message));
    };
   ```

3. Register the mobile device with the push notification service.

   ```
   MFPPush.registerDevice(function(successResponse) {
        alert("Successfully registered");
           },
         function(failureResponse) {
        alert("Failed to register");
           }
       );
   ```

4. (Optional) Unregister the mobile device from the push notification service.

   ```
   MFPPush.unregisterDevice(function(successResponse) {
        alert("Successfully unregistered");
           },
         function(failureResponse) {
        alert("Failed to unregister");
           }
       );
   ```

5. Remove WL.Client.Push.isPushSupported() (if used) and use.

   ```
   MFPPush.isPushSupported (function(successResponse) {
        alert(successResponse);
           },
         function(failureResponse) {
        alert("Failed to get the push suport status");
           }
       );
   ```

6. Remove the following WL.Client.Push APIs:
   a. onReadyToSubscribe()
   b. onMessage()

Server

Remove WL.Server.sendMessage() (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.

2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.

3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 4: Existing applications using tag notifications in their application:

Client

1. Initialize the MFPPush client instance in your application and in the success callback register the callback method that should receive the notification.
   ```
   MFPPush.initialize(function(successResponse){
        MFPPush.registerNotificationsCallback(notificationReceived);                  },
        function(failureResponse){
         alert("Failed to initialize");
        }
     );
   ```
2. Implement the notification callback method.
   ```
   var notificationReceived = function(message) {
         alert(JSON.stringify(message));
    };
   ```
3. Register the mobile device with the push notification service.
   ```
    MFPPush.registerDevice(function(successResponse) {
        alert("Successfully registered");
           },
         function(failureResponse) {
        alert("Failed to register");
           }
      );
   ```
4. (Optional) Un-register the mobile device from push notification service.
   ```
   MFPPush.unregisterDevice(function(successResponse) {
        alert("Successfully unregistered");
           },
         function(failureResponse) {
        alert("Failed to unregister");
           }
      );
   ```
5. Remove WL.Client.Push.isPushSupported() (if used) and use:
   ```
    MFPPush.isPushSupported (function(successResponse) {
        alert(successResponse);
           },
         function(failureResponse) {
        alert("Failed to get the push suport status");
           }
      );
   ```
6. Remove the following WL.Client.Push APIs:
   a. subscribeTag()
   b. unsubscribeTag()
   c. isTagSubscribed()
   d. onReadyToSubscribe()
   e. onMessage()
7. Subscribe to tags:

```
var tags = ['sample-tag1','sample-tag2']
       MFPPush.subscribe(tags, function(successResponse) {
     alert("Successfully subscribed");
        },
      function(failureResponse) {
     alert("Failed to subscribe");
        }
    );
```

8. (Optional) Unsubscribe from tags:

```
 MFPPush.unsubscribe(tags, function(successResponse) {
     alert("Successfully unsubscribed");
        },
      function(failureResponse) {
     alert("Failed to unsubscribe");
        }
    );
```

Server

Remove the following WL.Server.sendMessage() (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Native Android applications:

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application:

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in V8.0.0, convert this model to Unicast notification.

1. Initialize the MFPPush client instance in your application.

```
MFPPush push = MFPPush.getInstance();
          push.initialize(_this);
```

2. Implement the interface MFPPushNotificationListener and define onReceive().

```
@Override
    public void onReceive(MFPSimplePushNotification message) {
        Log.i("Push Notifications", message.getAlert());
    }
```

3. Register the mobile device with the push notification service.

```
  push.registerDevice(new MFPPushResponseListener<String>(){
                   @Override
                    public void onFailure(MFPPushException arg0) {
                        Log.i("Push Notifications", "Failed to register");
  }
                    @Override
                    public void onSuccess(String arg0) {
                       Log.i("Push Notifications", "Registered successfully");

                    }

                });
```

4. (Optional) Un-register the mobile device from the push notification service.

```
 push.unregisterDevice(new MFPPushResponseListener<String>(){
                    @Override
                    public void onFailure(MFPPushException arg0) {
                        Log.i("Push Notifications", "Failed to unregister");

                    }
                    @Override
                    public void onSuccess(String arg0) {
                         Log.i("Push Notifications", "Unregistered successfully");
                    }

                });
```

5. Remove WLClient.Push.isPushSupported() (if used) and use push.isPushSupported();.
6. Remove the following WLClient.Push APIs since there will be no event source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. WLOnReadyToSubscribeListener and WLNotificationListener implementation

Server

Remove the following WL.Server APIs (if used) in your adapter:

- notifyAllDevices()
- notifyDevice()
- notifyDeviceSubscription()
- createEventSource()

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.

2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 2: Existing applications using multiple event sources in their application:

Applications using multiple event sources requires the segmentation of users based on the subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this in MobileFirst V8.0.0, convert this model to tag based notification.

1. Initialize the MFPPush client instance in your application:
```
MFPPush push = MFPPush.getInstance();
          push.initialize(_this);
```
2. Implement the interface MFPPushNotificationListener and define onReceive().
```
 @Override
    public void onReceive(MFPSimplePushNotification message) {
        Log.i("Push Notifications", message.getAlert());

    }
```
3. Register the mobile device with the push notification service.
```
 push.registerDevice(new MFPPushResponseListener<String>(){
                   @Override
                   public void onFailure(MFPPushException arg0) {
                       Log.i("Push Notifications", "Failed to register");
  }
                   @Override
                   public void onSuccess(String arg0) {
                       Log.i("Push Notifications", "Registered successfully");

                   }

               });
```
4. (Optional) Un-register the mobile device from the push notification service:
```
  push.unregisterDevice(new MFPPushResponseListener<String>(){
                  @Override
                   public void onFailure(MFPPushException arg0) {
                       Log.i("Push Notifications", "Failed to unregister");

                   }
                   @Override
                   public void onSuccess(String arg0) {
                       Log.i( "Push Notifications", "Unregistered successfully");
                   }

               });
```
5. Remove WLClient.Push.isPushSupported() (if used) and use push.isPushSupported();.
6. Remove the following WLClient.Push APIs since there will be no event source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()

   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. WLOnReadyToSubscribeListener and WLNotificationListener Implementation
7. Subscribe to tags:
   ```
   String[] tags = new String[2];
                   tags[0] ="sample-tag1";
                   tags[1] ="sample-tag2";
       push.subscribe(tags, new MFPPushResponseListener<String[]>(){

                       @Override
                       public void onFailure(MFPPushException arg0) {
                           Log.i("Failed to subscribe");
                       }

                       @Override
                       public void onSuccess(String[] arg0) {
                           Log.i( "Subscribed successfully");
                       }
                   });
   ```
8. (Optional) Unsubscribe from tags:
   ```
    String[] tags = new String[2];
                   tags[0] ="sample-tag1";
                   tags[1] ="sample-tag2";
    push.unsubscribe(tags, new MFPPushResponseListener<String[]>(){

                       @Override
                       public void onFailure(MFPPushException arg0) {
                           Log.i("Push Notifications", "Failed to unsubscribe");

                       }

                       @Override
                       public void onSuccess(String[] arg0) {
                           Log.i("Push Notifications", "Unsubscribed successfully");

                       }

                   });
   ```

Server

Remove the following WL.Server APIs (if used) in your adapter:
- notifyAllDevices()
- notifyDevice()
- notifyDeviceSubscription()
- createEventSource()

Complete the following steps for every application that was using the same event source:
1. Set up the credentials by using the MobileFirst Operations Console. See “Configuring push notification settings” on page 7-259.

   You can also set up the credentials by using “Update GCM settings (PUT)” on page 8-193 REST API, for Android applications or “Update APNs settings (PUT)” on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.

3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 3: Existing applications using broadcast/Unicast notification in their application:

Client

1. Initialize the MFPPush client instance in your application:
   ```
   MFPPush push = MFPPush.getInstance();
             push.initialize(_this);
   ```
2. Implement the interface MFPPushNotificationListener and define onReceive().
   ```
   @Override
       public void onReceive(MFPSimplePushNotification message) {
           Log.i("Push Notifications", message.getAlert());

       }
   ```
3. Register the mobile device with push notification service.
   ```
   push.registerDevice(new MFPPushResponseListener<String>(){
                       @Override
                       public void onFailure(MFPPushException arg0) {
                           Log.i("Push Notifications", "Failed to register");
     }
                       @Override
                       public void onSuccess(String arg0) {
                           Log.i("Push Notifications", "Registered successfully");

                       }

                   });
   ```
4. (Optional) Un-register the mobile device from push notification service.
   ```
    push.unregisterDevice(new MFPPushResponseListener<String>(){
                      @Override
                       public void onFailure(MFPPushException arg0) {
                           Log.i("Push Notifications", "Failed to unregister");

                       }
                       @Override
                       public void onSuccess(String arg0) {
                           Log.i( "Push Notifications", "Unregistered successfully");
                       }

                   });
   ```
5. Remove `WLClient.Push.isPushSupported()` (if used) and use push.isPushSupported();.
6. Remove the following WLClient.Push APIs:
   - registerEventSourceCallback()
   - WLOnReadyToSubscribeListener and WLNotificationListener Implementation

Server

Remove WL.Server.sendMessage() (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 4: Existing applications using tag notifications in their application:

Client

1. Initialize the MFPPush client instance in your application:

```
MFPPush push = MFPPush.getInstance();
          push.initialize(_this);
```

2. Implement the interface MFPPushNotificationListener and define onReceive().

```
 @Override
    public void onReceive(MFPSimplePushNotification message) {
        Log.i("Push Notifications", message.getAlert());

    }
```

3. Register the mobile device with the push notification service.

```
    push.registerDevice(new MFPPushResponseListener<String>(){
                    @Override
                    public void onFailure(MFPPushException arg0) {
                        Log.i("Push Notifications", "Failed to register");
  }
                    @Override
                    public void onSuccess(String arg0) {
                        Log.i("Push Notifications", "Registered successfully");

                    }

                });
```

4. (Optional) Un-register the mobile device from the push notification service.

```
 push.unregisterDevice(new MFPPushResponseListener<String>(){
                    @Override
                    public void onFailure(MFPPushException arg0) {
                        Log.i("Push Notifications", "Failed to unregister");

                    }
                    @Override
                    public void onSuccess(String arg0) {
                        Log.i( "Push Notifications", "Unregistered successfully");
                    }

                });
```

5. Remove WLClient.Push.isPushSupported() (if used) and use push.isPushSupported();
6. Remove the following WLClient.Push API's:
   a. subscribeTag()
   b. unsubscribeTag()

c. isTagSubscribed()
d. WLOnReadyToSubscribeListener and WLNotificationListener Implementation

7. Subscribe to tags:

```
String[] tags = new String[2];
                tags[0] ="sample-tag1";
                tags[1] ="sample-tag2";
  push.subscribe(tags, new MFPPushResponseListener<String[]>(){

                    @Override
                    public void onFailure(MFPPushException arg0) {
                        Log.i("Failed to subscribe");
                    }

                    @Override
                    public void onSuccess(String[] arg0) {
                        Log.i( "Subscribed successfully");
                    }
                });
```

8. (Optional) Unsubscribe from tags:

```
String[] tags = new String[2];
                tags[0] ="sample-tag1";
                tags[1] ="sample-tag2";
     push.unsubscribe(tags, new MFPPushResponseListener<String[]>(){

                    @Override
                    public void onFailure(MFPPushException arg0) {
                        Log.i("Push Notifications", "Failed to unsubscribe");

                    }

                    @Override
                    public void onSuccess(String[] arg0) {
                        Log.i("Push Notifications", "Unsubscribed successfully");

                    }
  });
```

Server

Remove WL.Server.sendMessage() (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Native iOS applications:

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application:

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in V8.0.0, convert this model to Unicast notification.

1. Initialize the MFPPush client instance in your application.
   ```
   [[MFPPush sharedInstance] initialize];
   ```
2. Implement the notification processing in the didReceiveRemoteNotification().
3. Register the mobile device with the push notification service.
   ```
   [[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to register");
   }else{

    NSLog(@"Successfullyregistered");



   }
   }];
   ```
4. (Optional) Un-register the mobile device from the push notification service.
   ```
    [MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to unregister");
   }else{

    NSLog(@"Successfully unregistered");



   }
   }];
   ```
5. Remove WLClient.Push.isPushSupported() (if used) and use:
   ```
   [[MFPPush sharedInstance] isPushSupported]
   ```
6. Remove the following WLClient.Push API's since there will be no event source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. WLOnReadyToSubscribeListener implementation
7. Call sendDeviceToken() in didRegisterForRemoteNotificationsWithDeviceToken.
   ```
   [[MFPPush sharedInstance] sendDeviceToken:deviceToken];
   ```

Server

1. Remove the following WL.Server API's (if used) in your adapter:
   - notifyAllDevices()
   - notifyDevice()
   - notifyDeviceSubscription()
   - createEventSource()

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 2: Existing applications using multiple event sources in their application:

Applications using multiple event sources requires segmentation of users based on subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this to MobileFirstV8.0.0, convert this model to tag based notification.

1. Initialize the MFPPush client instance in your application.
   ```
   [[MFPPush sharedInstance] initialize];
   ```
2. Implement the notification processing in the didReceiveRemoteNotification().
3. Register the mobile device with the push notification service:
   ```
   [[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to register");
   }else{

    NSLog(@"Successfullyregistered");



   }
   }];
   ```
4. (Optional) Un-register the mobile device from the push notification service:
   ```
    [MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to unregister");
   ```

```
}else{

 NSLog(@"Successfully unregistered");



}
}];
```

5. Remove WLClient.Push.isPushSupported() (if used) and use:

   ```
   [[MFPPush sharedInstance] isPushSupported]
   ```

6. Remove the following WLClient.Push API's since there will be no event source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. WLOnReadyToSubscribeListener Implementation
7. Call sendDeviceToken() in didRegisterForRemoteNotificationsWithDeviceToken.
8. Subscribe to tags:

```
NSMutableArray *tags = [[NSMutableArray alloc]init];
    [tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
              [MFPPush sharedInstance] subscribe:tags completionHandler:^(WLResponse *response, NSError *error) {

if(error){

 NSLog(@"Failed to unregister");
}else{

 NSLog(@"Successfully unregistered");



}
}];
```

9. (Optional) Unsubscribe from tags:

```
 NSMutableArray *tags = [[NSMutableArray alloc]init];
    [tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
              [MFPPush sharedInstance] unsubscribe:tags completionHandler:^(WLResponse *response, NSError *error) {

if(error){

 NSLog(@"Failed to unregister");
}else{

 NSLog(@"Successfully unregistered");



}
}];
```

Server

1. Remove the following WL.Server API's (if used) in your adapter:
   - notifyAllDevices()
   - notifyDevice()
   - notifyDeviceSubscription()

- createEventSource()

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 3: Existing applications using broadcast/Unicast notification in their application:

Client

1. Initialize the MFPPush client instance in your application:
   ```
   [[MFPPush sharedInstance] initialize];
   ```
2. Implement the notification processing in the didReceiveRemoteNotification().
3. Register the mobile device with the push notification service:
   ```
   [[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to register");
   }else{

    NSLog(@"Successfullyregistered");



   }
   }];
   ```
4. (Optional) Un-register the mobile device from the push notification service.
   ```
   [MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to unregister");
   }else{

    NSLog(@"Successfully unregistered");



   }
   }];
   ```
5. Remove WLClient.Push.isPushSupported() (if used) and use:
   ```
   [[MFPPush sharedInstance] isPushSupported]
   ```
6. Remove the following WLClient.Push API's:
   - registerEventSourceCallback()
   - WLOnReadyToSubscribeListener Implementation

Server

Remove WL.Server.sendMessage (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See "Configuring push notification settings" on page 7-259.

   You can also set up the credentials by using "Update GCM settings (PUT)" on page 8-193 REST API, for Android applications or "Update APNs settings (PUT)" on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See "Creating tags for push notification" on page 7-260.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See "Sending push notifications to subscribers" on page 7-262.
   - The "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`.

Scenario 4: Existing applications using tag notifications in their application:

Client

1. Initialize the MFPPush client instance in your application:
   ```
   [[MFPPush sharedInstance] initialize];
   ```
2. Implement the notification processing in the didReceiveRemoteNotification().
3. Register the mobile device with the push notification service:
   ```
   [[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to register");
   }else{

    NSLog(@"Successfullyregistered");



   }
   }];
   ```
4. (Optional) Un-register the mobile device from the push notification service:
   ```
    [MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {

   if(error){

    NSLog(@"Failed to unregister");
   }else{

    NSLog(@"Successfully unregistered");



   }
   }];
   ```
5. Remove WLClient.Push.isPushSupported() (if used) and use [[MFPPush sharedInstance] isPushSupported].

6. Remove the following WLClient.Push API's since there will be no Event source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()
   b. subscribeTag()
   c. unsubscribeTag()
   d. isTagSubscribed()
   e. WLOnReadyToSubscribeListener Implementation
7. Call sendDeviceToken() in didRegisterForRemoteNotificationsWithDeviceToken.
8. Subscribe to tags:

```
  NSMutableArray *tags = [[NSMutableArray alloc]init];
    [tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
              [MFPPush sharedInstance] subscribe:tags completionHandler:^(WLResponse *response, NSError *error) {

if(error){

 NSLog(@"Failed to unregister");
}else{

 NSLog(@"Successfully unregistered");



}
}];
```

9. (Optional) Unsubscribe from tags:

```
 NSMutableArray *tags = [[NSMutableArray alloc]init];
    [tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
              [MFPPush sharedInstance] unsubscribe:tags completionHandler:^(WLResponse *response, NSError *error) {

if(error){

 NSLog(@"Failed to unregister");
}else{

 NSLog(@"Successfully unregistered");



}
}];
```

Server

Remove the WL.Server.sendMessage (if used), in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See “Configuring push notification settings” on page 7-259.

   You can also set up the credentials by using “Update GCM settings (PUT)” on page 8-193 REST API, for Android applications or “Update APNs settings (PUT)” on page 8-191 REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See “Creating tags for push notification” on page 7-260.

4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See “Sending push notifications to subscribers” on page 7-262.
   - The “Push Message (POST)” on page 8-236 REST API with `userId/deviceId`.

Native Windows Universal applications:

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application:

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in V8.0.0, convert this model to Unicast notification.

1. Initialize the MFPPush client instance in your application.
```
MFPPush push = MFPPush.GetInstance();
push.Initialize();
```
2. Implement the interface MFPPushNotificationListener and define onReceive().
```
class Pushlistener : MFPPushNotificationListener
{
    public void onReceive(String properties, String payload)
    {
            Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" +
    }
}
```
3. Register the mobile device with the push notification service.
```
MFPPushMessageResponse Response = await push.RegisterDevice(null);
if (Response.Success == true)
{
    Debug.WriteLine("Push Notifications Registered successfully");
}
else
{
    Debug.WriteLine("Push Notifications Failed to register");
}
```
4. (Optional) Un-register the mobile device from the push notification service.
```
MFPPushMessageResponse Response = await push.UnregisterDevice();
if (Response.Success == true)
{
    Debug.WriteLine("Push Notifications Failed to unregister");
}
else
{
    Debug.WriteLine("Push Notifications Unregistered successfully");
}
```
5. Remove WLClient.Push.IsPushSupported() (if used) and use push.IsPushSupported();.
6. Remove the following WLClient.Push APIs since there will be no event source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()

d. isSubscribed()
e. WLOnReadyToSubscribeListener and WLNotificationListener implementation

Server

Remove the following WL.Server APIs (if used) in your adapter:

- notifyAllDevices()
- notifyDevice()
- notifyDeviceSubscription()
- createEventSource()

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements** to **security checks** section in the **Security** tab of MobileFirst Operations Console.
3. You can also use the "Push Message (POST)" on page 8-236 REST API with `userId/deviceId`, to send message.

Scenario 2: Existing applications using multiple event sources in their application:

Applications using multiple event sources requires segmentation of users based on subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this in MobileFirst V8.0.0, convert this model to tag based notification.

1. Initialize the MFPPush client instance in your application:
   ```
   MFPPush push = MFPPush.GetInstance();
   push.Initialize();
   ```
2. Implement the interface MFPPushNotificationListener and define onReceive().
   ```
   class Pushlistener : MFPPushNotificationListener
   {
       public void onReceive(String properties, String payload)
       {
               Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" + pay
       }
   }
   ```
3. Register the mobile device with the IMFPUSH service.
   ```
   MFPPushMessageResponse Response = await push.RegisterDevice(null);
   if (Response.Success == true)
   {
       Debug.WriteLine("Push Notifications Registered successfully");
   }
   else
   {
       Debug.WriteLine("Push Notifications Failed to register");
   }
   ```
4. (Optional) Un-register the mobile device from the IMFPUSH service:
   ```
   MFPPushMessageResponse Response = await push.UnregisterDevice();
   if (Response.Success == true)
   {
   ```

```
    Debug.WriteLine("Push Notifications Failed to unregister");
}
else
{
    Debug.WriteLine("Push Notifications Unregistered successfully");
}
```

5. Remove WLClient.Push.IsPushSupported() (if used) and use push.IsPushSupported();.
6. Remove the following WLClient.Push APIs since there will be no Event Source to subscribe to and register notification callbacks:
   a. registerEventSourceCallback()
   b. subscribe()
   c. unsubscribe()
   d. isSubscribed()
   e. WLOnReadyToSubscribeListener and WLNotificationListener implementation
7. Subscribe to tags:

```
String[] Tag = { "sample-tag1", "sample-tag2" };
MFPPushMessageResponse Response = await push.Subscribe(Tag);
if (Response.Success == true)
{
    Debug.WriteLine("Subscribed successfully");
}
else
{
    Debug.WriteLine("Failed to subscribe");
}
```

8. (Optional) Unsubscribe from tags:

```
 String[] Tag = { "sample-tag1", "sample-tag2" };
MFPPushMessageResponse Response = await push.Unsubscribe(Tag);
if (Response.Success == true)
{
    Debug.WriteLine("Unsubscribed successfully");
}
else
{
    Debug.WriteLine("Failed to unsubscribe");
}
```

Server

Remove the following WL.Server APIs (if used) in your adapter:
- notifyAllDevices()
- notifyDevice()
- notifyDeviceSubscription()
- createEventSource()

Complete the following steps for every application that was using the same event source:
1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements** to **security checks** section in the **Security** tab of MobileFirst Operations Console.
3. Create Push tags in the **Tags** page of MobileFirst Operations Console.

4. You can also use the "Push Message (POST)" on page 8-236 REST API with `userId/deviceId/tagNames` as target, to send notifications.

Scenario 3: Existing applications using broadcast/Unicast notification in their application:

Client

1. Initialize the MFPPush client instance in your application:
   ```
   MFPPush push = MFPPush.GetInstance();
   push.Initialize();
   ```
2. Implement the interface MFPPushNotificationListener and define onReceive().
   ```
   class Pushlistener : MFPPushNotificationListener
   {
       public void onReceive(String properties, String payload)
       {
               Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" + pay
       }
   }
   ```
3. Register the mobile device with the push notification service.
   ```
   MFPPushMessageResponse Response = await push.RegisterDevice(null);
   if (Response.Success == true)
   {
       Debug.WriteLine("Push Notifications Registered successfully");
   }
   else
   {
       Debug.WriteLine("Push Notifications Failed to register");
   }
   ```
4. (Optional) Un-register the mobile device from the push notification service.
   ```
   MFPPushMessageResponse Response = await push.UnregisterDevice();
   if (Response.Success == true)
   {
       Debug.WriteLine("Push Notifications Failed to unregister");
   }
   else
   {
       Debug.WriteLine("Push Notifications Unregistered successfully");
   }
   ```
5. Remove `WLClient.Push.isPushSupported()` (if used) and use push.IsPushSupported();.
6. Remove the following WLClient.Push APIs:
   - registerEventSourceCallback()
   - WLOnReadyToSubscribeListener and WLNotificationListener implementation

Server

Remove WL.Server.sendMessage() (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements** to **security checks** section in the **Security** tab of MobileFirst Operations Console.
3. Create push tags in the **Tags** page of MobileFirst Operations Console.
4. You can also use the "Push Message (POST)" on page 8-236 REST API with `userId/deviceId/tagNames` as target, to send notifications.

Scenario 4: Existing applications using tag notifications in their application:

Client

1. Initialize the MFPPush client instance in your application:

   ```
   MFPPush push = MFPPush.GetInstance();
   push.Initialize();
   ```

2. Implement the interface MFPPushNotificationListener and define onReceive().

   ```
   class Pushlistener : MFPPushNotificationListener
   {
       public void onReceive(String properties, String payload)
       {
               Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" +
       }
   }
   ```

3. Register the mobile device with the push notification service.

   ```
   MFPPushMessageResponse Response = await push.RegisterDevice(null);
   if (Response.Success == true)
   {
       Debug.WriteLine("Push Notifications Registered successfully");
   }
   else
   {
       Debug.WriteLine("Push Notifications Failed to register");
   }
   ```

4. (Optional) Un-register the mobile device from push notification service.

   ```
   MFPPushMessageResponse Response = await push.UnregisterDevice();
   if (Response.Success == true)
   {
       Debug.WriteLine("Push Notifications Failed to unregister");
   }
   else
   {
       Debug.WriteLine("Push Notifications Unregistered successfully");
   }
   ```

5. Remove WLClient.Push.IsPushSupported() (if used) and use push.IsPushSupported();
6. Remove the following WLClient.Push API's:
   a. subscribeTag()
   b. unsubscribeTag()
   c. isTagSubscribed()
   d. WLOnReadyToSubscribeListener and WLNotificationListener implementation
7. Subscribe to tags:

   ```
   String[] Tag = { "sample-tag1", "sample-tag2" };
   MFPPushMessageResponse Response = await push.Subscribe(Tag);
   if (Response.Success == true)
   {
       Debug.WriteLine("Subscribed successfully");
   }
   else
   {
       Debug.WriteLine("Failed to subscribe");
   }
   ```

8. (Optional) Unsubscribe from tags:

   ```
    String[] Tag = { "sample-tag1", "sample-tag2" };
   MFPPushMessageResponse Response = await push.Unsubscribe(Tag);
   if (Response.Success == true)
   {
   ```

```
        Debug.WriteLine("Unsubscribed successfully");
    }
    else
    {
        Debug.WriteLine("Failed to unsubscribe");
    }
```

Server

Remove WL.Server.sendMessage() (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements** to **security checks** section in the **Security** tab of MobileFirst Operations Console.
3. Create push tags in the **Tags** page of MobileFirst Operations Console.
4. You can also use the "Push Message (POST)" on page 8-236 REST API with `userId/deviceId/tagNames` as target, to send notifications.

Migrating apps storing mobile data in Cloudant with IMFData or Cloudant SDK

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 Java , Objective-C, and Swift.

CloudantToolkit and IMFData frameworks are discontinued in IBM MobileFirst Platform Foundation V8.0.0.

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

Use the Cloudant Sync Android SDK as a replacement for CloudantToolkit and IMFData frameworks. With Cloudant Sync, you can persist data locally and replicate with a remote data store.

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 7-134.

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 7-265.

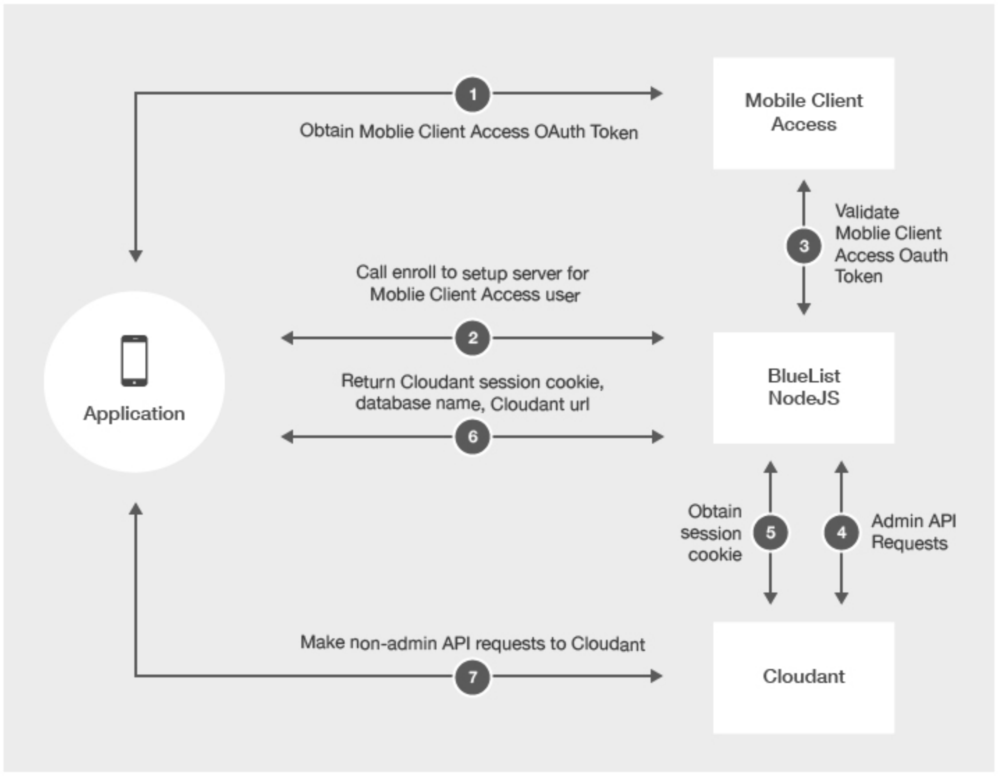
The Bluelist adapter sample has two primary functions:
- Exchange MobileFirst OAuth tokens for Cloudant session cookies
- Perform the required `admin` requests to Cloudant from the Bluelist sample.

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 Token
Mobile Client Access
3
Validate Moblie Client Access Oauth Token
Call enroll to setup server for Moblie Client Access user
2
Return Cloudant session cookie, database name, Cloudant url
6
Application
BlueList NodeJS
Obtain session cookie
5
4
Admin API Requests
Make non-admin API requests to Cloudant
7
Cloudant

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
}
```

```
// Get reference to DataManager
DataManager manager = DataManager.getInstance();

// Create local store
String name = "automobiledb";

Task<Store> storeTask = manager.localStore(name);
storeTask.continueWith(new Continuation<Store, Void>() {
    @Override
    public Void then(Task<Store> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
            // Do something with Store
            Store store = task.getResult();
        }
        return null;
    }
});
```

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
}
```

```
// Create DatastoreManager
       File path = context.getDir("databasedir", Context.MODE_PRIVATE);
       DatastoreManager manager = new DatastoreManager(path.getAbsolutePath());

       // Create a Datastore
       String name = "automobiledb";
       Datastore datastore = manager.openDatastore(name);
```

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 in
    if nil != error {
        //Handle error
    } else {
        let store:CDTStore = createdStore
        print("Successfully created store: \(store.name)")
    }
})
```

```
// Get reference to DataManager
DataManager manager = DataManager.getInstance();

// Create remote store
String name = "automobiledb";

Task<Store> storeTask = manager.remoteStore(name);
storeTask.continueWith(new Continuation<Store, Void>() {
    @Override
    public Void then(Task<Store> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
            // Do something with Store
            Store store = task.getResult();
        }
        return null;
    }
});
```

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: @"passw0rd" forIdentifier: @"identifier"];

//Initialize local store
CDTStore *localStore = [manager localStore: name withEncryptionKeyProvider: keyProvider error: &error];
```

```
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: @"passw0rd" forIdentifier: @"identifier"];

//Create local store
CDTDatastore *datastore = [datastoreManager datastoreNamed:name withEncryptionKeyProvider:keyProvider error:&error];
```

```
// 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: "identifier")
do{
    datastore = try datastoreManager.datastoreNamed(name, withEncryptionKeyProvider: keyProvider)
}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:@"password" forIdentifier:@"identifier"];

// pull replication
CDTPullReplication *pull = [manager pullReplicationForStore: databaseName withEncryptionKeyProvider: keyProvider];


// push replication
CDTPushReplication *push = [manager pushReplicationForStore: databaseName withEncryptionKeyProvider: keyProvider];
```

```
//Get reference to data manager
let manager = IMFDataManager.sharedInstance()
let databaseName = "automobiledb"

// Initalize a key provider
let keyProvider = CDTEncryptionKeychainProvider(password: "password", forIdentifier: "identifier")

// pull replication
let pull:CDTPullReplication = manager.pullReplicationForStore(databaseName, withEncryptionKeyProvider: keyProvider)


// push replication
let push:CDTPushReplication = manager.pushReplicationForStore(databaseName, withEncryptionKeyProvider: keyProvider)
```

AFTER (with Cloudant Sync):

Replication with an encrypted database requires no changes from replication with an unencrypted database.

Encrypting data on Android devices:

To encrypt data on an Android device, obtain encryption capabilities by including the correct libraries in your application. Then, you can initialize your local store for encryption and replicate data.

Procedure

1. Add the Cloudant Toolkit library as a dependency in your `build.gradle` file:
   BEFORE (with IMFData/CloudantToolkit):
   ```
   repositories {
       mavenCentral()
   }

   dependencies {
       compile 'com.ibm.mobile.services:cloudant-toolkit-local:1.0.0'
   }
   ```
   AFTER (with Cloudant Sync):

```
repositories {
    mavenLocal()
    maven { url "http://cloudant.github.io/cloudant-sync-eap/repository/" }
    mavenCentral()
}

dependencies {
    compile group: 'com.cloudant', name: 'cloudant-sync-datastore-core', version:'0.13.2'
    compile group: 'com.cloudant', name: 'cloudant-sync-datastore-android', version:'0.13.2'
    compile group: 'com.cloudant', name: 'cloudant-sync-datastore-android-encryption', version:'0.13.2'
}
```

2. Download the SQLCipher for Android v3.2 `.jar` and `.so` binary files and include them in your application in the appropriate folders within your app structure:
   a. Add libraries. Add the shared library files and SQLCipher archive to the `jniLibs` folder under your Android app directory.
   b. Add the required ICU compressed file to the `assets` folder in your app.

c. Add `sqlcipher.jar` as a file dependency. From the app folder menu in Android studio, select the **Dependencies** tab under **Open Module Settings**.

3. Initialize your local store for encryption with a key provider.

   Note: If you change the password after you create the database, an error occurs because the existing database cannot be decrypted. You cannot change your password after the database is encrypted. You must delete the database to change passwords.

   BEFORE (with IMFData/CloudantToolkit):

```
// Get reference to DataManager
DataManager manager = DataManager.getInstance();

// Initalize a key provider
KeyProvider keyProvider = new AndroidKeyProvider(getContext(),"password","identifier");

// Create local store
String databaseName = "automobiledb";
Task<Store> storeTask = manager.localStore(databaseName, keyProvider);
storeTask.continueWith(new Continuation<Store, Void >() {
    @Override
    public Void then(Task<Store> task) throws Exception {
        if (task.isFaulted()) {
            // Handle error
        } else {
            // Do something with Store
            Store store = task.getResult();
        }
        return null;
     }
});
```

   AFTER (with Cloudant Sync):

```
// Load SQLCipher libs
SQLiteDatabase.loadLibs(context);

// Create DatastoreManager
File path = context.getDir("databasedir", Context.MODE_PRIVATE);
DatastoreManager manager = new DatastoreManager(path.getAbsolutePath());

// Create encrypted local store
String name = "automobiledb";

KeyProvider keyProvider = new AndroidKeyProvider(context,"passw0rd","identifier");
Datastore datastore = manager.openDatastore(name, keyProvider);
```

4. When you are replicating data with an encrypted local store, you must pass a KeyProvider object into the pullReplicationForStore() or pushReplicationForStore() method.

   BEFORE (with IMFData/CloudantToolkit):

```
//Get reference to data manager
DataManager manager = DataManager.getInstance();
String databaseName = "automobiledb";

// Initalize a key provider
KeyProvider keyProvider = new AndroidKeyProvider(getContext(),"password","identifier");

// pull replication
Task<PushReplication> pullTask = manager.pullReplicationForStore(databaseName, keyProvider);

// push replication
Task<PushReplication> pushTask = manager.pushReplicationForStore(databaseName, keyProvider);
```

   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
    }
}
```

```
Task<Boolean> permissionsTask = manager.setCurrentUserPermissions(DataManager.DB_ACCESS_GROUP_MEMBERS, "automobiledb");

permissionsTask.continueWith(new Continuation<Boolean, Object>() {
    @Override
    public Object then(Task<Boolean> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
           // setting permissions was successful
        }
        return null;
    }
});
```

AFTER (with Cloudant Sync): There is no longer a way to 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 Mobile Client Access OAuth tokens with Cloudant Security, see the Bluelist sample.
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.

Cloudant stores data as JSON documents. The CloudantToolkit API provided an object mapper to map between native objects and JSON documents. Cloudant Sync does not provide this feature. The snippets in the following sections demonstrate how to use DocumentRevision 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)")
    }
})
```

```
// Use an existing store
Store store = existingStore;

// Create your object to save
Automobile automobile = new Automobile("Toyota", "Corolla", 2006);
```

```
// Save automobile to store
Task<Object> saveTask = store.save(automobile);
saveTask.continueWith(new Continuation<Object, Void>() {
    @Override
    public Void then(Task<Object> task) throws Exception {
        if (task.isFaulted()) {
            // save was not successful, task.getError() contains the error
        } else {
            // use the result
            Automobile savedAutomobile = (Automobile) task.getResult();
        }
        return null;
    }
});
```

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
}
```

```
// Use an existing store
Datastore datastore = existingStore;

// Create document body
Map<String, Object> body = new HashMap<String, Object>();
body.put("@datatype", "Automobile");
body.put("make", "Toyota");
body.put("model", "Corolla");
body.put("year", 2006);

// Create revision and set body
MutableDocumentRevision revision  = new MutableDocumentRevision();
revision.body = DocumentBodyFactory.create(body);

// Save revision to store
DocumentRevision savedRevision = datastore.createDocumentFromRevision(revision);
```

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)")
    }
})
```

```
// Use an existing store and documentId
Store store = existingStore;
String automobileId = existingAutomobileId;

// Fetch the automobile from the store
Task<Object> fetchTask = store.fetchById(automobileId);
fetchTask.continueWith(new Continuation<Object, Void>() {
    @Override
    public Void then(Task<Object> task) throws Exception {
        if (task.isFaulted()) {
            // fetch was not successful, task.getError() contains the error
        } else {
            // use the result
            Automobile fetchedAutomobile = (Automobile) task.getResult();
        }
        return null;
    }
});
```

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
}
```

```
// Use an existing store and documentId
Datastore datastore = existingStore;
String documentId = existingDocumentId;

// Fetch the revision from the store
DocumentRevision fetchedRevision = datastore.getDocument(documentId);
```

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)")
    }
})
```

```
// Use an existing store and Automobile
Store store = existingStore;
Automobile automobile = existingAutomobile;

// Update some of the values in the Automobile
automobile.setYear(2015);

// Save automobile to store
Task<Object> saveTask = store.save(automobile);
saveTask.continueWith(new Continuation<Object, Void>() {
    @Override
    public Void then(Task<Object> task) throws Exception {
        if (task.isFaulted()) {
            // save was not successful, task.getError() contains the error
        } else {
            // use the result
            Automobile savedAutomobile = (Automobile) task.getResult();
        }
        return null;
    }
});
```

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
}
```

```
// Use an existing store and documentId
// Use an existing store
Datastore datastore = existingStore;

// Make a MutableDocumentRevision from the existing revision
MutableDocumentRevision revision = existingRevision.mutableCopy();

// Update some of the values in the revision
Map<String, Object> body = revision.getBody().asMap();
```

```
body.put("year", 2015);
revision.body = DocumentBodyFactory.create(body);

// Save revision to store
DocumentRevision savedRevision = datastore.updateDocumentFromRevision(revision);
```

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)")
    }
})
```

```
// Use an existing store and automobile
Store store = existingStore;
Automobile automobile = existingAutomobile;

// Delete the automobile from the store
Task<String> deleteTask = store.delete(automobile);
deleteTask.continueWith(new Continuation<String, Void>() {
    @Override
    public Void then(Task<String> task) throws Exception {
        if (task.isFaulted()) {
            // delete was not successful, task.getError() contains the error
        } else {
            // use the result
            String deletedAutomobileId = task.getResult();
        }
        return null;
    }
});
```

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
}
```

```
// Use an existing store and revision
Datastore datastore = existingStore;
BasicDocumentRevision documentRevision = (BasicDocumentRevision) existingDocumentRevision;

// Delete revision from store
DocumentRevision deletedRevision = datastore.deleteDocumentFromRevision(documentRevision);
```

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.

For more details, see Cloudant Sync Query documentation. For CRUD operations on a remote store, see Cloudant's 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
    }
}
```

```
// Use an existing data store
Store store = existingStore;

// The data type to use for the Automobile class
String dataType = store.getMapper().getDataTypeForClassName(Automobile.class.getCanonicalName());

// The fields to index.
List<IndexField> indexFields = new ArrayList<IndexField>();
indexFields.add(new IndexField("year"));
indexFields.add(new IndexField("make"));

// Create the index
Task<Void> indexTask = store.createIndexWithDataType(dataType, indexFields);
indexTask.continueWith(new Continuation<Void, Void>() {
    @Override
    public Void then(Task<Void> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
            // Continue application flow
        }
        return null;
    }
});
```

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
}
```

```
// Use an existing store
Datastore datastore = existingStore;

// Create an IndexManager
IndexManager indexManager = new IndexManager(datastore);

// The fields to index.
List<Object> indexFields = new ArrayList<Object>();
indexFields.add("@datatype");
indexFields.add("year");
indexFields.add("make");

// Create the index
indexManager.ensureIndexed(indexFields, "automobile_index");
```

- 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
    }
})
```

```
// Use an existing data store
Store store = existingStore;
String indexName = existingIndexName;

// Delete the index
Task<Void> indexTask = store.deleteIndex(indexName);
indexTask.continueWith(new Continuation<Void, Void>() {
    @Override
    public Void then(Task<Void> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
            // Continue application flow
        }
        return null;
    }
});
```

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
}
```

```
// Use an existing store
Datastore datastore = existingStore;
String indexName = existingIndexName;
IndexManager indexManager = existingIndexManager;

// Delete the index
indexManager.deleteIndexNamed(indexName);
```

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 more details, see Cloudant Sync 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])] withP

[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
}
```

- Create and run a query for objects on Android.

To run a query for objects, create a Cloudant query with the query filters on data type. Run the query against a Store object.

BEFORE (with IMFData/CloudantToolkit):

```
// Use an existing store
Store store = existingStore;

// Create data type predicate
Map<String, Object> dataTypeEqualityOpMap = new HashMap<String, Object>();
dataTypeEqualityOpMap.put("$eq", "Automobile");

Map<String, Object> dataTypeSelectorMap = new HashMap<String, Object>();
dataTypeSelectorMap.put("@datatype", dataTypeEqualityOpMap);

// Create year predicate
Map<String, Object> yearEqualityOpMap = new HashMap<String, Object>();
yearEqualityOpMap.put("$eq", 2006);

Map<String, Object> yearSelectorMap = new HashMap<String, Object>();
yearSelectorMap.put("year", yearEqualityOpMap);

// Add predicates to AND compound predicate
List<Map<String, Object>> andPredicates = new ArrayList<Map<String, Object>>();
andPredicates.add(dataTypeSelectorMap);
andPredicates.add(yearSelectorMap);

Map<String, Object> andOpMap = new HashMap<String, Object>();
andOpMap.put("$and", andPredicates);

Map<String, Object> cloudantQueryMap = new HashMap<String, Object>();
cloudantQueryMap.put("selector", andOpMap);

// Create a Cloudant Query Object
CloudantQuery query = new CloudantQuery(cloudantQueryMap);

// Run the Cloudant Query against a Store
Task<List> queryTask = store.performQuery(query);
queryTask.continueWith(new Continuation<List, Object>() {
    @Override
    public Object then(Task<List> task) throws Exception {
        if(task.isFaulted()){
            // Handle Error
        }else{
            List queryResult = task.getResult();
            // Use queryResult to do something
        }
        return null;
    }
});
```

AFTER (with Cloudant Sync):

```
// Use an existing store
Datastore datastore = existingStore;
IndexManager indexManager = existingIndexManager;

// Create data type predicate
Map<String, Object> dataTypeEqualityOpMap = new HashMap<String, Object>();
dataTypeEqualityOpMap.put("$eq", "Automobile");

Map<String, Object> dataTypeSelectorMap = new HashMap<String, Object>();
dataTypeSelectorMap.put("@datatype", dataTypeEqualityOpMap);

// Create year predicate
Map<String, Object> yearEqualityOpMap = new HashMap<String, Object>();
yearEqualityOpMap.put("$eq", 2006);

Map<String, Object> yearSelectorMap = new HashMap<String, Object>();
```

```
  yearSelectorMap.put("year", yearEqualityOpMap);

  // Add predicates to AND compound predicate
  List<Map<String, Object>> andPredicates = new ArrayList<Map<String, Object>>();
  andPredicates.add(dataTypeSelectorMap);
  andPredicates.add(yearSelectorMap);

  Map<String, Object> selectorMap = new HashMap<String, Object>();
  selectorMap.put("$and", andPredicates);

  // Run the query against a Store
  QueryResult result = indexManager.find(selectorMap);
```

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 more details, see Cloudant Sync 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
}
```

```
// Use an existing store
Store store = existingStore;

// create a pull replication task
// name is the database name of the store being replicated
Task<PullReplication> pullTask = manager.pullReplicationForStore(store.getName());
pullTask.continueWith(new Continuation<PullReplication, Object>() {
    @Override
    public Object then(Task<PullReplication> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
            // Start the replication
            PullReplication pull = task.getResult();
            Replicator replicator = ReplicatorFactory.oneway(pull);
            replicator.start();
        }
        return null;
    }
});
```

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
}
```

```
// Use an opened Datastore to replicate to
Datastore datastore = existingDatastore;
URI uri = existingURI;

// Create a replicator that replicates changes from the remote
final Replicator replicator = ReplicatorBuilder.pull().from(uri).to(datastore).build();

// Register event listener
replicator.getEventBus().register(new Object() {

    @Subscribe
    public void complete(ReplicationCompleted event) {

        // Handle ReplicationCompleted event
    }

    @Subscribe
    public void error(ReplicationErrored event) {

        // Handle ReplicationErrored event
    }
});

// Start replication
replicator.start();
```

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
}
```

```
// Use an existing store
Store store = existingStore;

// create a push replication task
// name is the database name of the store being replicated
Task<PushReplication> pushTask = manager.pushReplicationForStore(store.getName());
pushTask.continueWith(new Continuation<PushReplication, Object>() {
    @Override
    public Object then(Task<PushReplication> task) throws Exception {
        if(task.isFaulted()){
            // Handle error
        }else{
            // Start the replication
            PushReplication push = task.getResult();
            Replicator replicator = ReplicatorFactory.oneway(push);
            replicator.start();
        }
        return null;
    }
});
```

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
}
```

```
// Use an opened Datastore to replicate from
Datastore datastore = existingStore;
URI uri = existingURI;

// Create a replicator that replicates changes from the local
// database to the remote datastore.
final Replicator replicator = ReplicatorBuilder.push().from(datastore).to(uri).build();

// Register event listener
replicator.getEventBus().register(new Object() {

    @Subscribe
    public void complete(ReplicationCompleted event) {

        // Handle ReplicationCompleted event
    }

    @Subscribe
    public void error(ReplicationErrored event) {

        // Handle ReplicationErrored event
    }
});

// Start replication
replicator.start();
```

Applying a fix pack to IBM MobileFirst Platform Server

Find out how to use the Server Configuration Tool to upgrade MobileFirst Server V8.0.0 to a fix pack or an interim fix. Alternatively, if you installed MobileFirst Server with Ant tasks, you can also use Ant tasks to apply the fix pack or interim fix.

About this task

To apply an interim fix or fix pack on MobileFirst Server, choose one of the following topics based on your initial installation method:

- Applying a fix pack or an interim fix with the Server Configuration Tool
- "Applying a fix pack by using the Ant files" on page 6-111

Installing and configuring server-side components

Learn how to install and configure the server-side components of IBM MobileFirst Platform Foundation.

To learn how to install and configure the server-side components of IBM MobileFirst Platform Foundation, read the following topics.

For more information about how to set up a development environment, see "Setting up the development environment" on page 7-9.

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

Installation overview

IBM MobileFirst Platform Foundation provides development tools and server-side components that you can install on-premises or deploy to the cloud for test or production use. Review the installation topics appropriate for your installation scenario.

Installing a development environment

If you develop the client-side or the server-side of mobile apps, install MobileFirst Platform CLI, MobileFirst Development Server and MobileFirst development libraries for your environment. For more information, see "Setting up the development environment" on page 7-9.

Installing a test or production server on-premises

If you install a test or production server, start with "Tutorials about MobileFirst Server installation" on page 6-4 for a simple installation and to learn about the installation of MobileFirst Server. For more information about preparing an installation for your specific environment, see "Installing MobileFirst Server for a production environment" on page 6-39.

To add MobileFirst Analytics Server to your installation, see "MobileFirst Analytics Server installation guide" on page 11-2.

To install IBM MobileFirst Platform Application Center, see "Installing and configuring the Application Center" on page 6-198.

Deploying MobileFirst Server to the cloud

If you plan to deploy MobileFirst Server to the cloud, see "Deploying MobileFirst Server to the cloud" on page 9-1

Upgrading from earlier versions

The preceding sections provide an overview of IBM MobileFirst Platform Foundation new installations. For information about upgrading existing installations and applications to a newer version, see "Upgrading to IBM

MobileFirst Platform Foundation V8.0.0" on page 5-1.

Installing IBM MobileFirst Platform Server

IBM installations are based on an IBM product called IBM Installation Manager. Install IBM Installation Manager V1.8.4 or later separately before you install IBM MobileFirst Platform Foundation.

Important: Ensure that you use IBM Installation Manager V1.8.4 or later. The older versions of Installation Manager are not able to install IBM MobileFirst Platform Foundation V8.0.0 because the postinstallation operations of the product require Java 7. The older versions of Installation Manager come with Java 6.

The MobileFirst Server installer copies onto your computer all the tools and libraries that are required for deploying MobileFirst Server components and optionally the IBM MobileFirst Platform Application Center to your application server.

The following topics include an overview of MobileFirst Server architecture, a getting started tutorial, and the complete information about the installation of MobileFirst Server for production.

MobileFirst Server overview

MobileFirst Server consists of several components. An overview of MobileFirst Server architecture is provided for you to understand the functions of each component.

Unlike MobileFirst Server V7.1.0 or earlier, the installation process for V8.0.0 is separated from the development and deployment of mobile app operations. In V8.0.0, after the server components and the database are installed and configured, MobileFirst Server can be operated for most operations without the need to access the application server or database configuration.

The administration and deployment operations of the MobileFirst artifacts are done through MobileFirst Operations Console, or the REST API of the MobileFirst Server administration service. The operations can also be done by using some command line tools that wrap this API, such as `mfpdev` or `mfpadm`. The authorized users of MobileFirst Server can modify the server-side configuration of mobile applications, upload, or configure server-side code (the adapters), upload new web resources for Cordova mobile apps, run application management operations, and more.

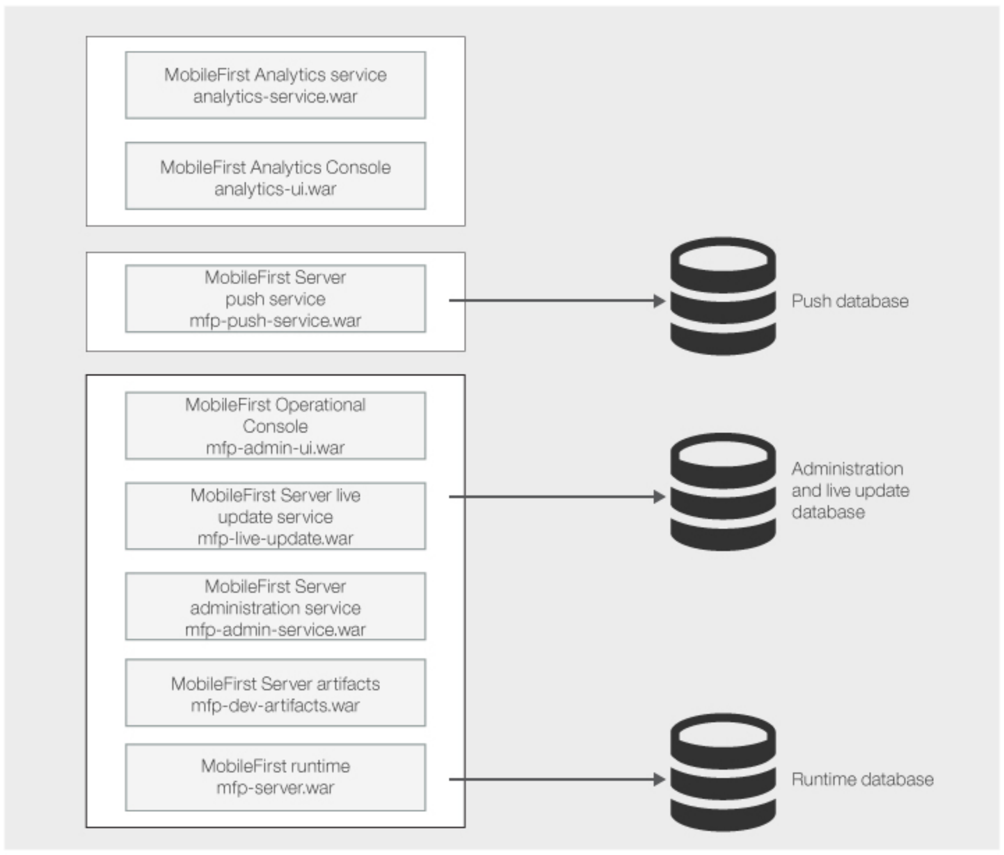
MobileFirst Server offers extra layers of security, in addition to the security layers of the network infrastructure or the application server. The security features include the control of application authenticity and the access control to the server-side resources and the adapters. These security configurations can also be done by the authorized users of MobileFirst Operations Console and the administration service. You determine the authorization of the MobileFirst administrators by mapping them to security roles as described in "Configuring user authentication for MobileFirst Server administration" on page 6-166.

A simplified version of MobileFirst Server that is preconfigured and does not need software prerequisite such as database or an application server is available for developers. See "Setting up the MobileFirst Development Server" on page 7-12.

For production purpose, you can use an installation of MobileFirst Server on-premises or deploy MobileFirst Server to the cloud. For more information about deploying MobileFirst Server to the cloud, see “Deploying MobileFirst Server to the cloud” on page 9-1.

MobileFirst Server components

The architecture of the MobileFirst Server components is illustrated as follows:


MobileFirst Analytics service analytics-service.war
MobileFirst Analytics Console analytics-ui.war
MobileFirst Server push service mfp-push-service.war
Push database
MobileFirst Operational Console mfp-admin-ui.war
MobileFirst Server live update service mfp-live-update.war
Administration and live update database
MobileFirst Server administration service mfp-admin-service.war
MobileFirst Server artifacts mfp-dev-artifacts.war
MobileFirst runtime mfp-server.war
Runtime database

MobileFirst Server is composed of several components.

Core components of MobileFirst Server

MobileFirst Operations Console, the MobileFirst Server administration service, the MobileFirst Server live update service, the MobileFirst Server artifacts, and the MobileFirst runtime are the minimum set of the components to install. The runtime provides the MobileFirst services to the mobile apps that run on the mobile devices. The administration service provides the configuration and administration capabilities. You use the service via MobileFirst Operations Console, the live update service REST API, or command line tools such as `mfpadm` or `mfpdev`. The live update service manages configuration data and is used by the administration service. These components require a database. The database table name for each component does not have any intersection. As such, you can use the same database or even the same schema to store all the tables of these components. For more information, see “Setting up databases” on page 6-63. It is possible to install more than one instance of the runtime. In this case, each instance needs its own database. The artifacts component provides resources for MobileFirst Operations Console. It does not requires a database.

Optional components of MobileFirst Server

The MobileFirst Server push service provides push notification capabilities. It must be installed to provide these capabilities of the mobile apps use the MobileFirst Push features ("Push notification" on page 7-248). From the perspective of mobile apps, the URL of the push service is the same as the URL as the runtime, except that its context root is `/imfpush`. If you plan to install the push service on a different server or cluster than the runtime, you need to configure the routing rules of your HTTP server. The configuration is to ensure that the requests to the push service and the runtime are properly routed. The push service requires a database. The tables of the push service have no intersection with the tables of the runtime, the administration service, and the live update service. Thus, it can also be installed in the same database or schema. The MobileFirst Analytics service and MobileFirst Analytics Console provide monitoring and analytics information about the mobile apps usage. Mobile apps can provide more insight by using the "Logger SDK" on page 11-37. The MobileFirst Analytics service does not need a database. It stores its data locally on disk by using Elasticsearch. The data is structured in shards that can be replicated between the members of a cluster of the Analytics service.

For more information about the network flows and the topology constraints for these components, see "Topologies and network flows" on page 6-78.

Installation process

The installation of MobileFirst Server on-premises can be done by using the following ways:

- The Server Configuration Tool - a graphical wizard
- Ant tasks through the command line tools
- Manual installation

In the next sections, more information about the installation of MobileFirst Server on-premises is provided. You can find:

- A getting started tutorial that guides you through a complete installation of MobileFirst Server farm on WebSphere Application Server Liberty profile. The tutorial is based on a simple scenario for you to try out the installation either in graphical mode or in command line mode.
- A detailed section ("Installing MobileFirst Server for a production environment" on page 6-39) that contains details about the installation prerequisites, database setup, server topologies, deployment of the components to the application server, and server configuration.

Tutorials about MobileFirst Server installation

Learn about the MobileFirst Server installation process by walking through the instructions to create a functional MobileFirst Server, cluster with two nodes on WebSphere Application Server Liberty profile.

This getting started tutorial guides you through the installation procedure to have a functional MobileFirst Server, clusters with two nodes on Liberty profile. The installation can be done in two ways:

- By using the graphical mode of IBM Installation Manager and the Server Configuration Tool.
- By using the command line tool.

After the tutorial is completed, you have a working MobileFirst Server. However, you need to configure it, in particular for security, before you use the server. For more information, see "Configuring MobileFirst Server" on page 6-164.

Installing MobileFirst Server in graphical mode

Use the graphical mode of IBM Installation Manager and the Server Configuration Tool to install MobileFirst Server.

Before you begin

1. Make sure that one of the following databases and a supported Java version are installed. You also need the corresponding JDBC driver for the database to be available on your computer:
   - Database Management System (DBMS) from the list of supported database:
     - DB2®
     - MySQL
     - Oracle

     Important:

     You must have a database where you can create the tables that are needed by the product, and a database user who can create tables in that database.

     In the tutorial, the steps to create the tables are for DB2. You can find the DB2 installer as a package of IBM MobileFirst Platform Foundation eAssembly on IBM Passport Advantage.
   - JDBC driver for your database.
     - For DB2, use the DB2 JDBC driver type 4.
     - For MySQL, use the Connector/J JDBC driver.
     - For Oracle, use the Oracle thin JDBC driver.
   - Java 7 or later.
2. Download the installer of IBM Installation Manager V1.8.4 or later from Installation Manager and Packaging Utility download links.
3. You must also have the installation repository of the MobileFirst Server and the installer of WebSphere Application Server Liberty Core V8.5.5.3 or later. Download these packages from the IBM MobileFirst Platform Foundation eAssembly on Passport Advantage:

   MobileFirst Server installation repository
   IBM MobileFirst Platform Foundation V8.0 `.zip` file of Installation Manager Repository for IBM MobileFirst Platform Server

   WebSphere Application Server Liberty profile
   IBM WebSphere Application Server - Liberty Core V8.5.5.3 or later

About this task

You can run the installation in graphical mode if you are on one of the following operating systems:

- Windows x86 or x86-64
- Mac OS x86-64
- Linux x86 or Linux x86-64

On other operating systems, you can still run the installation with Installation Manager in graphical mode, but the Server Configuration Tool is not available. You need to use Ant tasks (as described in “Installing MobileFirst Server in command line mode” on page 6-22) to deploy MobileFirst Server to Liberty profile.

Note: The instruction to install and set up the database is not part of this tutorial. If you want to run this tutorial without installing a stand-alone database, you can use the embedded Derby database. However, the restrictions for using this database are as follows:

- You can run Installation Manager in graphical mode, but to deploy the server, you need to skip to the command line section of this tutorial to install with Ant tasks.
- You cannot configure a server farm. Embedded Derby database does not support access from multiple servers. To configure a server farm, you need DB2, MySQL, or Oracle.

This tutorial goes through the following steps:

1. “Installing IBM Installation Manager”
2. “Installing WebSphere Application Server Liberty Core”
3. “Installing MobileFirst Server” on page 6-8
4. “Creating a database” on page 6-9
5. “Running the Server Configuration Tool” on page 6-10
6. “Testing the installation” on page 6-18
7. “Creating a farm of two Liberty servers that run MobileFirst Server” on page 6-19
8. “Testing the farm and see the changes in MobileFirst Operations Console” on page 6-22

Installing IBM Installation Manager:
About this task

You must install Installation Manager V1.8.4 or later. The older versions of Installation Manager are not able to install IBM MobileFirst Platform Foundation V8.0 because the postinstallation operations of the product require Java 7. The older versions of Installation Manager come with Java 6.

Procedure

1. Extract the IBM Installation Manager archive that is downloaded. You can find the installer at Installation Manager and Packaging Utility download links.
2. Install Installation Manager.
   - Run `install.exe` to install Installation Manager as administrator. Root is needed on Linux or UNIX. On Windows, the administrator privilege is needed. In this mode, the information about the installed packages is placed in a shared location on the disk and any user that is allowed to run Installation Manager can update the applications.
   - Run `userinst.exe` to install Installation Manager in user mode. No specific privilege is needed. However, in this mode, the information about the installed packages are placed in the user's home directory. Only that user can update the applications that are installed with Installation Manager.

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. In this task, Liberty profile is installed and a server instance is created so that you can install MobileFirst Server on it.

Procedure

1. Extract the compressed file for WebSphere Application Server Liberty Core that you downloaded.
2. Launch Installation Manager.
3. Add the repository in Installation Manager.
   a. Go to **File** > **Preferences** and click **Add Repositories...**.
   b. Browse for the `repository.config` file of `diskTag.inf` file in the directory where the installer is extracted.
   c. Select the file and click **OK**.
   d. Click **OK** to close the **Preferences** panel.
4. Click **Install** to install Liberty.
   a. Select `IBM WebSphere Application Server Liberty Core` and click **Next**.
   b. Accept the terms in the license agreements, and click **Next**.
5. In the scope of this tutorial, do not need to install the additional assets when asked. Click **Install** for the installation process to start.
   - If the installation is successful, the program displays a message indicating that installation is successful. The program might also display important postinstallation instructions.
   - If the installation is not successful, click **View Log File** to troubleshoot the problem.
6. Move the `usr` directory that contains the servers in a location that does not need specific privileges.

   If you install Liberty with Installation Manager in administrator mode, the files are in a location where non-administrator or non-root users cannot modify the files. For the scope of this tutorial, move the `usr` directory that contains the servers in a place that does not need specific privileges. In this way, the installation operations can be done without specific privileges.
   a. Go to the installation directory of Liberty.
   b. Create a directory named `etc`. You need administrator or root privileges.
   c. In `etc` directory, create a `server.env` file with the following content:
   ```
   WLP_USER_DIR=<path to a directory where any user can write>
   ```
   For example, on Windows:
   ```
   WLP_USER_DIR=C:\LibertyServers\usr
   ```
7. Create a Liberty server that will be used to install the first node of MobileFirst Server at the later part of the tutorial.
   a. Start a command line.
   b. Go to *`liberty_install_dir`*`/bin`, and enter **`server create mfp1`**.

   This command creates a Liberty server instance named mfp1. You can see its definition at *`liberty_install_dir`*`/usr/servers/mfp1` or *`WLP_USER_DIR`*`/servers/mfp1` (if you modify the directory as described in step 6).

Results

After the server is created, you can start this server with **`server start mfp1`** from *`liberty_install_dir`*`/bin/`.

To stop the server, enter the command: **`server stop mfp1`** from *`liberty_install_dir`*`/bin/`.

The default home page can be viewed at `http://localhost:9080`.

Note: For production, you need to make sure that the Liberty server is started as a service when the host computer starts. Making the Liberty server start as a service is not part of this tutorial.

Installing MobileFirst Server:
Before you begin

Make sure that Installation Manager V1.8.4 or later is installed. The installation of MobileFirst Server might not succeed with an older version of Installation Manager because the postinstallation operations require Java 7. The older versions of Installation Manager come with Java 6.

About this task

Run Installation Manager to install the binary files of MobileFirst Server on your disk before you create the databases and deploy MobileFirst Server to Liberty profile. During the installation of MobileFirst Server with Installation Manager, an option is proposed to you to install IBM MobileFirst Platform Application Center. Application Center is a different component of the product. For this tutorial, it is not required to be installed with MobileFirst Server. For more information about Application Center, see "Installing and configuring the Application Center" on page 6-198.

Procedure
1. Launch Installation Manager.
2. Add the repository of MobileFirst Server in Installation Manager.
   a. Go to **File** > **Preferences** and click **Add Repositories...**.
   b. Browse for the repository file in the directory where the installer is extracted.

      If you decompress the IBM MobileFirst Platform Foundation V8.0 `.zip` file for MobileFirst Server in *`mfp_installer_directory`* folder, the repository file can be found at *`mfp_installer_directory`*`/MobileFirst_Platform_Server/disk1/diskTag.inf`.

      You might also want to apply the latest fix pack that can be downloaded from IBM Support Portal. Make sure to enter the repository for the fix pack. If you decompress the fix pack in *`fixpack_directory`* folder, the repository file is found in *`fixpack_directory`*`/MobileFirst_Platform_Server/disk1/diskTag.inf`.

      Note: You cannot install the fix pack without the repository of the base version in the repositories of Installation Manager. The fix packs are incremental installers and need the repository of the base version to be installed.
   c. Select the file and click **OK**.
   d. Click **OK** to close the **Preferences** panel.
3. After you accept the license terms of the product, click **Next**.
4. Select the `Create a new package group` option to install the product in that new package group.

5. Click **Next**.
6. Select `Do not activate token licensing with the Rational License Key Server` option in the **Activate token licensing** section of the **General settings** panel.

   In this tutorial, it is assumed that token licensing is not needed and the steps to configure MobileFirst Server for token licensing are not included. However, for production installation, you must determine whether you need to activate token licensing or not. If you have a contract to use token licensing with Rational License Key Server, select `Activate token licensing with the Rational License Key Server` option. After you activate token licensing, you must do extra steps to configure MobileFirst Server. For more information, see "Installing and configuring for token licensing" on page 6-150.
7. Keep the default option (`No`) as-is in the **Install IBM MobileFirst Platform Foundation for iOS** section of the **General settings** panel.
8. Select `No` option in the **Choose configuration** panel so that Application Center is not installed. For production installation, use Ant tasks to install Application Center. The installation with Ant tasks enables you to decouple the updates to MobileFirst Server from the updates to Application Center. For more information about installing Application Center, see "Installing and configuring the Application Center" on page 6-198.
9. Click **Next** until you reach the **Thank You** panel. Then, proceed with the installation.

Results

An installation directory that contains the resources to install MobileFirst components is installed.

You can find the resources in the following folders:
- `MobileFirstServer` folder for MobileFirst Server
- `PushService` folder for MobileFirst Server push service
- `ApplicationCenter` folder for Application Center
- `Analytics` folder for MobileFirst Analytics

The goal of this tutorial is to install MobileFirst Server by using the resources in `MobileFirstServer` folder.

You can also find some shortcuts for the Server Configuration Tool, Ant, and **`mfpadm`** program in the `shortcuts` folder.

Creating a database:
About this task

This task is to ensure that a database exists in your DBMS, and that a user is allowed to use the database, create tables in it, and use the tables.

The database is used to store the technical data that is used by the various MobileFirst components:
- MobileFirst Server administration service
- MobileFirst Server live update service
- MobileFirst Server push service
- MobileFirst runtime

In this tutorial, the tables for all the components are placed under the same schema. The Server Configuration Tool creates the tables in the same schema. For more flexibility, you might want to use Ant tasks or a manual installation.

Note: The steps in this task are for DB2. If you plan to use MySQL or Oracle, see "Database requirements" on page 6-65.

Procedure

1. Log on to the computer that is running the DB2 server. It is assumed that a DB2 user, for example named as **`mfpuser`**, exists.
2. Verify that this DB2 user has the access to a database with a page size 32768 or more, and is allowed to create implicit schemas and tables in that database.

   By default, this user is a user declared on the operating system of the computer that runs DB2. That is, a user with a login for that computer. If such user exists, the next action in step 3 is not needed. In the later part of the tutorial, the Server Configuration Tool creates all the tables that are required by the product under a schema in that database.
3. Create a database with the correct page size for this installation if you do not have one.
   a. Open a session with a user that has `SYSADM` or `SYSCTRL` permissions. For example, use the user **`db2inst1`** that is the default admin user that is created by the DB2 installer.
   b. Open a DB2 command line processor:
      - On Windows systems, click **Start** > **IBM DB2** > **Command Line Processor**.
      - On Linux or UNIX systems, go to `~/sqllib/bin` (or *`db2_install_dir`*`/bin` if `sqllib` is not created in the administrator's home directory) and enter `./db2`.

   c. Enter the following SQL statements to create a database that is called **`MFPDATA`**:
      ```
      CREATE DATABASE MFPDATA COLLATE USING SYSTEM PAGESIZE 32768
      CONNECT TO MFPDATA
      GRANT CONNECT ON DATABASE TO USER mfpuser
      GRANT CREATETAB ON DATABASE TO USER mfpuser
      GRANT IMPLICIT_SCHEMA ON DATABASE TO USER mfpuser
      DISCONNECT MFPDATA
      QUIT
      ```
      If you defined a different user name, replace **`mfpuser`** with your own user name.

      Note: The statement does not remove the default privileges granted to PUBLIC in a default DB2 database. For production, you might need to reduce the privileges in that database to the minimum requirement for the product. For more information about DB2 security and an example of the security practices, see DB2 security, Part 8: Twelve DB2 security best practices.

Running the Server Configuration Tool:
About this task

You use the Server Configuration Tool to run the following operations:
- Create the tables in the database that are needed by the MobileFirst applications

- Deploy the web applications of MobileFirst Server (the runtime, administration service, live update service, push service components, and MobileFirst Operations Console) to Liberty server.

The Server Configuration Tool does not deploy the following MobileFirst applications:

MobileFirst Analytics
: MobileFirst Analytics is typically deployed on a different set of servers than MobileFirst Server because of its high memory requirements. MobileFirst Analytics can be installed manually or with Ant tasks. If it is already installed, you can enter its URL, the user name, and password to send data to it in the Server Configuration Tool. The Server Configuration Tool will then configure the MobileFirst apps to send data to MobileFirst Analytics. For more information about the installation of MobileFirst Analytics, see "MobileFirst Analytics Server installation guide" on page 11-2.

Application Center
: This application can be used to distribute mobile apps internally to the employees that use the apps, or for test purpose. It is independent of MobileFirst Server and is not necessary to install together with MobileFirst Server. For more information, see "Installing and configuring the Application Center" on page 6-198.

Procedure

1. Start the Server Configuration Tool.
   - On Linux, from application shortcuts **Applications** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Windows, click **Start** > **Programs** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Mac OS, open a shell console. Go to `*mfp_server_install_dir*/shortcuts` and type `./configuration-tool.sh`.

     The *mfp_server_install_dir* directory is where you installed MobileFirst Server.
2. Select **File** > **New Configuration...** to create a MobileFirst Server Configuration.
3. Name the configuration `Hello MobileFirst` and click **OK**.
4. Leave the default entries of **Configuration Details** as-is and click **Next**.

   In this tutorial, the environment ID is not used. It is a feature for advanced deployment scenario. An example of such scenario would be installing multiple instances of MobileFirst Server and administration service in the same application server or WebSphere Application Server cell.
5. Keep the default context root for the administration service and the runtime component.
6. Do not change the default entries in the **Console Settings** panel and click **Next** to install MobileFirst Operations Console with the default context root.
7. Select `IBM DB2` as a database and click **Next**.
8. In the **DB2 Database Settings** panel, complete the details:
   a. Enter the host name that runs your DB2 server. If it is running on your computer, you can enter `localhost`.
   b. Change the port number if the DB2 instance you plan to use is not listening to the default port (50000).

c. Enter the path to the DB2 JDBC driver. For DB2, the file that is named as `db2jcc4.jar` is expected. It is also needed to have the `db2jcc_license_cu.jar` file in the same directory. In a standard DB2 distribution, these files are found in *`db2_install_dir`*`/java`.

d. Click **Next**.

If the DB2 server cannot be reached with the credentials that are entered, the Server Configuration Tool disables the **Next** button and displays an error. The **Next** button is also disabled if the JDBC driver does not contain the expected classes. If everything is correct, the **Next** button is enabled.

9. In the **DB2 Additional Settings** panel, complete the details:

   a. Enter `mfpuser` as DB2 user name and its password. Use your own DB2 user name if it is not **`mfpuser`**.

   b. Enter `MFPDATA` as the name of the database.

   c. Leave `MFPDATA` as the schema in which the tables will be created. Click **Next**. By default, Server Configuration Tool proposes the value `MFPDATA`.

10. Do not enter any values in the **Database Creation Request** panel and click **Next**.

    This pane is used when the database that is entered in the previous pane does not exist on the DB2 server. In that case, you can enter the user name and password of the DB2 administrator. The Server Configuration Tool opens an ssh session to the DB2 server and runs the commands as described in "Creating a database" on page 6-9 to create the database with default settings and the correct page size.

11. In the **Application Server Selection** panel, select `WebSphere Application Server` option and click **Next**.

12. In the **Application Server Settings** panel, complete the details:

    a. Enter the installation directory for WebSphere Application Server Liberty.

    b. Select the server where you plan to install the product in the server name field. Select `mfp1` server that is created in step 7 on page 6-7 of "Installing WebSphere Application Server Liberty Core" on page 6-6.

    c. Leave the `Create a user` option selected with its default values.

       This option creates a user in the basic registry of the Liberty server, so that you can sign in to MobileFirst Operations Console or to the administration service. For a production installation, do not use this option and configure the security roles of the applications after the installation as described in "Configuring user authentication for MobileFirst Server administration" on page 6-166.

    d. Select the `Server farm deployment` option for the deployment type.

    e. Click **Next**.

13. Select `Install the Push service` option.

    When the push service is installed, HTTP or HTTPS flows are needed from the administration service to the push service, and from the administration service and the push service to the runtime component.

14. Select `Have the Push and Authorization Service URLs computed automatically` option.

    When this option is selected, the Server Configuration Tool configures the applications to connect to the applications installed on the same server. When you use a cluster, enter the URL that is used to connect to the services from your HTTP load balancer. When you install on WebSphere Application Server Network Deployment, it is mandatory to enter a URL manually.

15. Keep the default entries of **Credentials for secure communication between the Administration and the Push service** as-is.

    A client ID and a password are needed to register the push service and the administration service as the confidential OAuth clients for the authorization server (which is by default, the runtime component). The Server Configuration Tool generates an ID and a random password for each of the service, that you can keep as-is for this getting started tutorial.
16. Click **Next**.
17. Keep the default entries of **Analytics Setting** panel as-is.

    To enable the connection to the Analytics server, you need to first install MobileFirst Analytics. However, the installation is not in the scope of this tutorial.
18. Click **Deploy**.

Results

You can see a detail of the operations done in **Console Window**.

An Ant file is saved. The Server Configuration Tool helps you create an Ant file for installing and updating your configuration. This Ant file can be exported by using **File** > **Export Configuration as Ant Files...** . For more information about this Ant file, see "Deploying MobileFirst Server to Liberty with Ant tasks" on page 6-28 in "Installing MobileFirst Server in command line mode" on page 6-22.

Then, the Ant file is run and does the following operations:
1. The tables for the following components are created in the database:
   - The administration service and the live update service. Created by the `admdatabases` Ant target.
   - The runtime. Created by the `rtmdatabases` Ant target.
   - The push service. Created by the `pushdatabases` Ant target.
2. The WAR files of the various components are deployed to Liberty server. You can see the details of the operations in the log under `adminstall`, `rtminstall`, and `pushinstall` targets.

If you have access to the DB2 server, you can list the tables that are created by using these instructions:
1. Open a DB2 command line processor with **`mfpuser`** as described in step 3 on page 6-10 of "Creating a database" on page 6-9.
2. Enter the SQL statements:
   ```
   CONNECT TO MFPDATA USER mfpuser USING mfpuser_password
   LIST TABLES FOR SCHEMA MFPDATA
   DISCONNECT MFPDATA
   QUIT
   ```

Take note of the following database factors:

Database user consideration

In the Server Configuration Tool, only one database user is needed. This user is used to create the tables, but is also used as the data source user in the application server at run time. In production environment, you might want to restrict the privileges of the user that is used at run time to the strict minimum (`SELECT / INSERT / DELETE / UPDATE`), and thus provide a different user for deployment in the application server. For more information about the privileges that are required at run time, see

"Database users and privileges" on page 6-64. The Ant files that are provided as examples also use the same users for both cases. However, in the case of DB2, you might want to create your own versions of files. As such, you can distinguish the user that is used to create the databases from the user that is used for the data source in the application server with the Ant tasks.

Database tables creation
For production, you might want to create the tables manually. For example, if your DBA wants to override some default settings or assign specific table spaces. The database scripts that are used to create the tables are available in *mfp_server_install_dir*/MobileFirstServer/databases and *mfp_server_install_dir*/PushService/databases. For more information, see "Create the database tables manually" on page 6-67.

The server.xml file and some application server setting are modified during the installation. Before each modification, a copy of the server.xml file is made, such as server.xml.bak, server.xml.bak1, and server.xml.bak2. To see everything that was added, you can compare the server.xml file with the oldest backup (server.xml.bak). On Linux, you can use the command **diff --strip-trailing-cr server.xml server.xml.bak** to see the differences. On AIX, use the command **diff server.xml server.xml.bak** to find the differences.

Modification of the application server settings (specific to Liberty):

1. The Liberty features are added.

   The features are added for each application and can be duplicated. For example, the JDBC feature is used for both the administration service and the runtime components. This duplication allows the removal of the features of an application when it is uninstalled without breaking the other applications. For example, if you decide at some point to uninstall the push service from a server and install it on another server. However, not all topologies are possible. The administration service, the live update service, and the runtime component must be on the same application server with Liberty profile. For more information, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84. The duplication of features does not create issue unless the features that added are conflicting. Adding the jdbc-40 and jdbc-41 features would cause a problem, but adding twice the same feature does not.
2. host='*' is added in the httpEndPoint declaration.

   This setting is to allow the connection to the server from all network interfaces. In production, you might want to restrict the host value of the HTTP endpoint.
3. The tcpOptions element (tcpOptions soReuseAddr="true") is added in the server configuration to enable immediate rebind to a port with no active listener and improve the throughput of the server.
4. A keystore with ID **defaultKeyStore** is created if it does not exist.

   The keystore is to enable the HTTPS port and more specifically, to enable the JMX communication between the administration service (mfp-admin-service.war) and the runtime component (mfp-server.war). The two applications communicate via JMX. In the case of Liberty profile, restConnector is used to communicate between the applications in a single server and also between the servers of a Liberty Farm. It requires the use of HTTPS. For the keystore that is created by default, Liberty profiles creates a certificate with a validity period of 365 days.

This configuration is not intended for production use. For production, you need to reconsider to use your own certificate.

To enable JMX, a user with administrator role (named as **MfpRESTUser**) is created in the basic registry. Its name and password are provided as JNDI properties (**mfp.admin.jmx.user** and **mfp.admin.jmx.pwd**) and are used by the runtime component and the administration service to run JMX queries. In the global JMX properties, some properties are used to define the cluster mode (stand-alone server or working in a farm). The Server Configuration Tool sets the **mfp.topology.clustermode** property to `Standalone` in Liberty server. In the later part of this tutorial about the creation of a farm, the property is modified to `Cluster`.

5. The creation of users (Also valid for Apache Tomcat and WebSphere Application Server)
   - Optional Users: The Server Configuration Tool creates a test user (`admin/admin`) so that you can use this user to log to the console after the installation.
   - Mandatory Users: The Server Configuration Tool also creates a user (named as **configUser_mfpadmin** with a randomly generated password) to be used by the administration service to contact the local live update service. For Liberty server, **MfpRESTUser** is created. If your application server is not configured to use a basic registry (for example, an LDAP registry), the Server Configuration Tool is unable to request the name of an existing user. In this case, you need to use Ant tasks. For more information, see “Installing with Ant Tasks” on page 6-110.
6. The `webContainer` element is modified.

   The `deferServletLoad` web container custom property is set to false. Both the runtime component and the administration service must start when the server starts. These components can thus register the JMX beans and start the synchronization procedure that allows the runtime component to download all the applications and adapters that it needs to serve.
7. The default executor is customized to set large values to **coreThreads** and **maxThreads** if you use Liberty V8.5.5.5 or earlier. The default executor is automatically tuned by Liberty as of V8.5.5.6.

   This setting avoids timeout issues that break the startup sequence of the runtime component and administration service on some Liberty versions. The absence of this statement can be the cause of these errors in the server log file:

```
Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: Read timed out
FWLSE3000E: A server error was detected.
FWLSE3012E: JMX configuration error. Unable to obtain MBeans. Reason: "Read timed out".
```

Declaration of applications

The following applications are installed:

- `mfpadmin`, the administration service
- `mfpadminconfig`, the live update service
- `mfpconsole`, MobileFirst Operations Console
- `mobilefirst`, MobileFirst runtime component
- `imfpush`, the push service

The Server Configuration Tool installs all the applications on the same server. You can separate the applications in different application servers, but under certain constraints that are documented in "Topologies and network flows" on page 6-78.

For an installation on different servers, you cannot use the Server Configuration Tool. Use Ant tasks ("Deploying MobileFirst Server to Liberty with Ant tasks" on page 6-28) or install the product manually.

Administration service

The administration service is the service for managing MobileFirst applications, adapters, and their configurations. It is secured by security roles. By default, the Server Configuration Tool adds a user (**`admin`**) with the administrator role, that you can use to log in to the console for testing. The configuration of the security role must be done after an installation with the Server Configuration Tool (or with Ant tasks). See "Configuring user authentication for MobileFirst Server administration" on page 6-166. You might want to map the users or the groups that come from the basic registry or an LDAP registry that you configure in your application server to each security role.

The class loader is set with delegation parent last for Liberty profile and WebSphere Application Server, and for all MobileFirst applications. This setting is to avoid conflicts between the classes packaged in the MobileFirst applications and the classes of the application server. Forgetting to set the class loader delegation to parent last is a frequent source of error in manual installation. For Apache Tomcat, this declaration is not needed.

In Liberty profile, a common library is added to the application for decrypting passwords that are passed as JNDI properties. The Server Configuration Tool defines two mandatory JNDI properties for the administration service: **`mfp.config.service.user`** and **`mfp.config.service.password`**. They are used by the administration service to connect to the live update service with its REST API. More JNDI properties can be defined to tune the application or adapt it to your installation particularities. For more information, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

The Server Configuration Tool also defines the JNDI properties (the URL and the OAuth parameters to register the confidential clients) for the communication with the push service.

The data source to the database that contains the tables for the administration service is declared, as well as a library for its JDBC driver.

Live update service

The live update service stores information about the runtime and application configurations. It is controlled by the administration service and must always run on the same server as the administration service. The context root is *`context_root_of_admin_server`*`config`. As such, it is `mfpadminconfig`. The administration service assumes that this convention is respected to create the URL of its requests to the REST services of the live update service.

The class loader is set with delegation parent last as discussed in the administration service section.

The live update service has one security role, **admin_config**. A user must be mapped to that role. Its password and login must be provided to the administration service with the JNDI property: **mfp.config.service.user** and **mfp.config.service.password**. For information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174 and "List of JNDI properties for MobileFirst Server live update service" on page 6-182.

It also needs a data source with JNDI name on Liberty profile. The convention is `*context_root_of_config_server*/jdbc/ConfigDS`. In this tutorial, it is defined as `mfpadminconfig/jdbc/ConfigDS`. In an installation by the Server Configuration Tool or with Ant tasks, the tables of the live update service are in the same database and schema as the tables of the administration service. The user to access these tables is also the same.

MobileFirst Operations Console

MobileFirst Operations Console is declared with the same security roles as the administration service. The users that are mapped to the security roles of MobileFirst Operations Console must also be mapped to the same security role of the administration service. Indeed, MobileFirst Operations Console runs queries to the administration service on the behalf of the console user.

The Server Configuration Tool positions one JNDI property, **mfp.admin.endpoint**, that indicates how the console connects to the administration service. The default value set by the Server Configuration Tool is `'*://*:*/mfpadmin'`. The setting means that it must use the same protocol, host name, and port as the incoming HTTP request to the console, and the context root of the administration service is `/mfpadmin`. If you want to force the request to go though a web proxy, change the default value. For more information about the possible values for this URL, or for information about other possible JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

The class loader is set with delegation parent last as discussed in the administration service section.

MobileFirst runtime

This application is not secured by a security role. It is not required to log in with a user known by the Liberty server, to access this application. The mobile devices requests are routed to the runtime. They are authenticated by other mechanisms specific to the product (such as OAuth) and the configuration of the MobileFirst applications.

The class loader is set with delegation parent last as discussed in the administration service section.

It also needs a data source with JNDI name on Liberty profile. The convention is `*context_root_of_runtime*/jdbc/mfpDS`. In this tutorial, it is defined as `mobilefirst/jdbc/mfpDS`. In an installation by the Server Configuration Tool or with Ant tasks, the tables of the runtime are in the same database and schema as the tables of the administration service. The user to access these tables is also the same.

Push service

This application is secured by OAuth. The valid OAuth tokens must be included in any HTTP request to the service.

The configuration of OAuth is made through the JNDI properties (such as the URL of the authorization server, the client ID, and the password of the push service). The JNDI properties also indicate the security plug-in (**`mfp.push.services.ext.security`**) and the fact that a relational database is used (**`mfp.push.db.type`**). The requests from the mobile devices to the push service are routed to this service. The context root of the push service must be `/imfpush`. The client SDK computes the URL of the push service based on the URL of the runtime with the context root (`/imfpush`). If you want to install the push service on a different server than the runtime, you need to have an HTTP router that can route the device requests to the relevant application server.

The class loader is set with delegation parent last as discussed in the administration service section.

It also needs a data source with JNDI name on Liberty profile. The JNDI name is `imfpush/jdbc/imfPushDS`. In an installation by the Server Configuration Tool or with Ant tasks, the tables of the push service are in the same database and schema as the tables of the administration service. The user to access these tables is also the same.

Other files modification

The Liberty profile `jvm.options` file is modified. A property (**`com.ibm.ws.jmx.connector.client.rest.readTimeout`**) is defined to avoid timeout issues with JMX when the runtime synchronizes with the administration service.

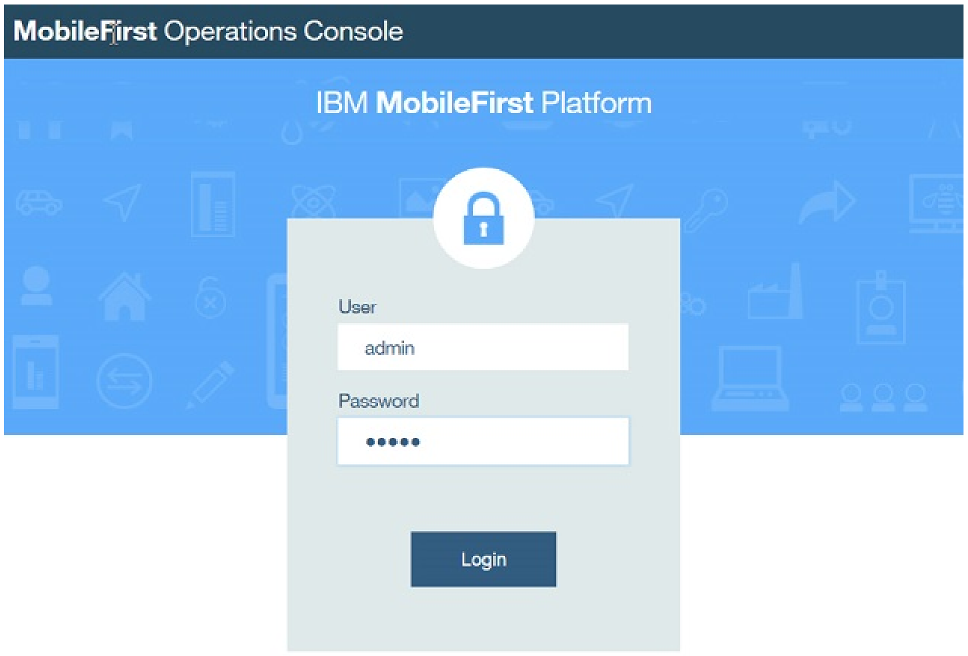
Testing the installation:
About this task

After the installation is complete, you can use this procedure to test the components that are installed.

Procedure

1. Start the server by using the command **`server start mfp1`**. The binary file for the server is in *`liberty_install_dir`*`/bin`.
2. Test MobileFirst Operations Console with a web browser.

   Go to `http://localhost:9080/mfpconsole`. By default, the server runs on port 9080. However, you can verify the port in the element `<httpEndpoint>` as defined in the `server.xml` file. A login screen is displayed.



3. Log in with `admin/admin`.

   This user is created by default by the Server Configuration Tool.

   Note: If you connect with HTTP, the login ID and password are sent in clear text in the network. For a secure login, use HTTPS to log to the server. You can see the HTTPS port of the Liberty server in the **`httpsPort`** attribute of the `<httpEndpoint>` element in the `server.xml` file. By default, the value is 9443.
4. Log out of the console with **Hello Admin** > **Sign Out**.
5. Enter the following URL: `https://localhost:9443/mfpconsole` in the web browser and accept the certificate.

   By default, the Liberty server generates a default certificate that is not known by your web browser, you need to accept the certificate. Mozilla Firefox presents this certification as a security exception.
6. Log in again with `admin/admin`.

   The login and password are encrypted between your web browser and MobileFirst Server. In production, you might want to close the HTTP port.

Creating a farm of two Liberty servers that run MobileFirst Server:
About this task

In this task, you will create a second Liberty server that runs the same MobileFirst Server and connected to the same database. In production, you might use more than one server for performance reasons, to have enough servers to serve the number of transactions per second that is needed for your mobile applications at peak time. It is also for high availability reasons to avoid having a single point of failure.

When you have more than one server that runs MobileFirst Server, the servers must be configured as a farm. This configuration enables any administration service to contact all the runtimes of a farm. If the cluster is not configured as a farm, only the runtime that runs in the same application server as the management service that runs the management operation is notified. Others runtimes are not aware of the change. For example, you deploy a new version of an adapter in a cluster that is not configured as a farm, only one server would serve the new adapter. The other servers would continue to serve the old adapter. The only situation where you can have a cluster and do not need to configure a farm is when you install your servers on WebSphere Application Server Network Deployment. The administration service is able to find all the servers by querying the JMX beans with the deployment manager. The deployment manager must be running to allow management operations because it is used to provide the list of the MobileFirst JMX beans of the cell.

When you create a farm, you also need to configure an HTTP server to send queries to all the members of the farm. The configuration of an HTTP server is not included in this tutorial. This tutorial is only about configuring the farm so that management operations are replicated to all the runtime components of the cluster.

Procedure

1. Create a second Liberty server on the same computer.
   a. Start a command line.
   b. Go to `*liberty_install_dir*/bin`, and enter **`server create mfp2`**.
2. Modify the HTTP and HTTPS ports of the server mfp2 so that they do not conflict with the ports of server mfp1.
   a. Go to the second server directory.

      The directory is `*liberty_install_dir*/usr/servers/mfp2` or `*WLP_USER_DIR*/servers/mfp2` (if you modify the directory as described in step 6 on page 6-7 of "Installing WebSphere Application Server Liberty Core" on page 6-6).
   b. Edit the `server.xml` file. Replace

      ```
      <httpEndpoint id="defaultHttpEndpoint"
          httpPort="9080"
          httpsPort="9443" />
      ```

      with

      ```
      <httpEndpoint id="defaultHttpEndpoint"
          httpPort="9081"
          httpsPort="9444" />
      ```

      The HTTP and HTTPS ports of the server mfp2 do not conflict with the ports of the server mfp1 with this change. Make sure to modify the ports before you run the installation of MobileFirst Server. Otherwise, if you modify the port after the installation is made, you also need to reflect the change of the port in the JNDI property: **`mfp.admin.jmx.port`**.
3. Run the Server Configuration Tool.
   a. Create a configuration `Hello MobileFirst 2`.
   b. Do the same installation procedure as described in "Running the Server Configuration Tool" on page 6-10 but select `mfp2` as the application server. Use the same database and same schema.

      Note:
      - If you use an environment ID for server mfp1 (not suggested in the tutorial), the same environment ID must be used for server mfp2.
      - If you modify the context root for some applications, use the same context root for server mfp2. The servers of a farm must be symmetric.

- If you create a default user (admin/admin), create the same user in the server mfp2.

The Ant tasks detect that the databases exist and do not create the tables (see the following log extract). Then, the applications are deployed to the server.

```
[configuredatabase] Checking connectivity to MobileFirstAdmin database MFPDATA with schema 'MFPDATA' and user 'mfpuser'...
[configuredatabase] Database MFPDATA exists.
[configuredatabase] Connection to MobileFirstAdmin database MFPDATA with schema 'MFPDATA' and user 'mfpuser' succeeded.
[configuredatabase] Getting the version of MobileFirstAdmin database MFPDATA...
[configuredatabase] Table MFPADMIN_VERSION exists, checking its value...
[configuredatabase] GetSQLQueryResult => MFPADMIN_VERSION = 8.0.0
[configuredatabase] Configuring MobileFirstAdmin database MFPDATA...
[configuredatabase] The database is in latest version (8.0.0), no upgrade required.
[configuredatabase] Configuration of MobileFirstAdmin database MFPDATA succeeded.
```

4. Test the two servers with HTTP connection.
   a. Open a web browser.
   b. Enter the following URL: `http://localhost:9080/mfpconsole`. The console is served by server mfp1.
   c. Log in with `admin/admin`.
   d. Open a tab in the same web browser and enter the URL: `http://localhost:9081/mfpconsole`. The console is served by server mfp2.
   e. Log in with `admin/admin`. If the installation is done correctly, you can see the same welcome page in both tabs after login.
   f. Return to first browser tab and click **Hello, admin** > **Download Audit Log**. You are logged out of the console and see the login screen again.

      This logout behavior is an issue. The problem happens because when you log on to server mfp2, a Lightweight Third Party Authentication (LTPA) token is created and stored in your browser as a cookie. However, this LTPA token is not recognized by server mfp1. Switching between servers is likely to happen in a production environment when you have an HTTP load balancer in front of the cluster. To resolve this issue, you must ensure that both servers (mfp1 and mfp2) generate the LTPA tokens with the same secret keys. Copy the LTPA keys from server mfp1 to server mfp2.

      1) Stop both servers with these commands:

         ```
         server stop mfp1
         server stop mfp2
         ```

      2) Copy the LTPA keys of server mfp1 to server mfp2.

         From *liberty_install_dir*/usr/servers or *WLP_USER_DIR*/servers, run the following command depending on your operating system.

         - On UNIX: **cp mfp1/resources/security/ltpa.keys mfp2/resources/security/ltpa.keys**
         - On Windows: **copy mfp1/resources/security/ltpa.keys mfp2/resources/security/ltpa.keys**

   g. Restart the servers. Switch from one browser tab to another other does not require you to relogin. In a Liberty server farm, all servers must have the same LTPA keys.

5. Enable the JMX communication between the Liberty servers.

   The JMX communication with Liberty, is done via the Liberty REST connector over the HTTPS protocol. To enable this communication, each server of the farm must be able to recognize the SSL certificate of the other members. You need to exchange the HTTPS certificates in their truststores. Use IBM utilities such as Keytool, which is part of the IBM JRE distribution in `java/bin` to configure the truststore. 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

WLP_USER_DIR/servers/*server_name*/resources/security/key.jks. The password of this default keystore, as can be seen in the server.xml file, is mobilefirst.

Tip: You can change it with the Keytool utility, but you must also change the password in the server.xml file so that Liberty server can read that keystore. In this tutorial, use the default password.

a. In *WLP_USER_DIR*/servers/mfp1/resources/security, enter **keytool -list -keystore key.jks**. The command shows the certificates in the keystore. There is only one named **default**. You are prompted for the password of the keystore (mobilefirst) before you can see the keys. This is the case for all the next commands with Keytool utility.
b. Export the default certificate of server mfp1 with the command: **keytool -exportcert -keystore key.jks -alias default -file mfp1.cert**.
c. In *WLP_USER_DIR*/servers/mfp2/resources/security, export the default certificate of server mfp2 with the command: **keytool -exportcert -keystore key.jks -alias default -file mfp2.cert**.
d. In the same directory, import the certificate of server mfp1 with the command: **keytool -import -file ../../../mfp1/resources/security/mfp1.cert -keystore key.jks** The certificate of server mfp1 is imported into the keystore of server mfp2 so that server mfp2 can trust the HTTPS connections to server mfp1. You are asked to confirm that you trust the certificate.
e. In *WLP_USER_DIR*/servers/mfp1/resources/security, import the certificate of server mfp2 with the command: **keytool -import -file ../../../mfp2/resources/security/mfp2.cert -keystore key.jks**. After this step, the HTTPS connections between the two servers are possible.

Testing the farm and see the changes in MobileFirst Operations Console:
Procedure

1. Start the two servers:
```
server start mfp1
server start mfp2
```
2. Access the console. For example, http://localhost:9080/mfpconsole, or https://localhost:9443/mfpconsole in HTTPS. In the left sidebar, an extra menu that is labeled as **Server Farm Nodes** appears. If you click **Server Farm Nodes**, you can the status of each node. You might need to wait a bit for both nodes to be started.

Installing MobileFirst Server in command line mode

Use the command line mode of IBM Installation Manager and Ant tasks to install MobileFirst Server.

Before you begin

1. Make sure that one of the following databases and a supported Java version are installed. You also need the corresponding JDBC driver for the database to be available on your computer:
   - Database Management System (DBMS) from the list of supported database:
     - DB2
     - MySQL
     - Oracle

   Important:

You must have a database where you can create the tables that are needed by the product, and a database user who can create tables in that database.

In the tutorial, the steps to create the tables are for DB2. You can find the DB2 installer as a package of IBM MobileFirst Platform Foundation eAssembly on IBM Passport Advantage.

- JDBC driver for your database.
  - For DB2, use the DB2 JDBC driver type 4.
  - For MySQL, use the Connector/J JDBC driver.
  - For Oracle, use the Oracle thin JDBC driver.
- Java 7 or later.

2. Download the installer of IBM Installation Manager V1.8.4 or later from Installation Manager and Packaging Utility download links.
3. You must also have the installation repository of the MobileFirst Server and the installer of WebSphere Application Server Liberty Core V8.5.5.3 or later. Download these packages from the IBM MobileFirst Platform Foundation eAssembly on Passport Advantage:

   MobileFirst Server installation repository
   IBM MobileFirst Platform Foundation V8.0 `.zip` file of Installation Manager Repository for IBM MobileFirst Platform Server

   WebSphere Application Server Liberty profile
   IBM WebSphere Application Server - Liberty Core V8.5.5.3 or later

About this task

This tutorial goes through the following steps:

1. "Installing IBM Installation Manager"
2. "Installing WebSphere Application Server Liberty Core" on page 6-24
3. "Installing MobileFirst Server" on page 6-25
4. "Creating a database" on page 6-27
5. "Deploying MobileFirst Server to Liberty with Ant tasks" on page 6-28
6. "Testing the installation" on page 6-35
7. "Creating a farm of two Liberty servers that run MobileFirst Server" on page 6-36
8. "Testing the farm and see the changes in MobileFirst Operations Console" on page 6-39

Installing IBM Installation Manager:
About this task

You must install Installation Manager V1.8.4 or later. The older versions of Installation Manager are not able to install IBM MobileFirst Platform Foundation V8.0 because the postinstallation operations of the product require Java 7. The older versions of Installation Manager come with Java 6.

Procedure

1. Extract the IBM Installation Manager archive file that is downloaded. You can find the installer at Installation Manager and Packaging Utility download links.
2. Review the license agreement for IBM Installation Manager that is in *`unzip_IM_1.8.x`*`/license` directory.

3. If you accept the license agreement after the review, install Installation Manager.
   - Run `installc.exe` to install Installation Manager as administrator. Root is needed on Linux or UNIX. On Windows, the administrator privilege is needed. In this mode, the information about the installed packages is placed in a shared location on the disk and any user that is allowed to run Installation Manager can update the applications.

     The executable file name ends with `c` (`installc`) for a command line installation without a graphical user interface. To install Installation Manager, enter **`installc.exe -acceptLicence`**.
   - Run `userinstc.exe` to install Installation Manager in user mode. No specific privilege is needed. However, in this mode, the information about the installed packages are placed in the user's home directory. Only that user can update the applications that are installed with Installation Manager.

     The executable ends with `c` (`userinstc`) for a command line installation without a graphical user interface. To install Installation Manager, enter **`userinstc.exe -acceptLicence`**.

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. In this task, Liberty profile is installed and a server instance is created so that you can install MobileFirst Server on it.

Procedure

1. Review the license agreement for WebSphere Application Server Liberty Core. The license files can be viewed when you download the installer from Passport Advantage.
2. Extract the compressed file of WebSphere Application Server Liberty Core, that you downloaded, to a folder.

   In the steps that follow, the directory where you extract the installer is referred as *`liberty_repository_dir`*. It contains a `repository.config` file or a `diskTag.inf` file, among many other files.
3. Decide a directory where Liberty profile is to be installed. It is referred as *`liberty_install_dir`* in the next steps.
4. Start a command line and go to *`installation_manager_install_dir`*`/tools/eclipse/`.
5. If you accept the license agreement after the review, install Liberty.

   Enter the command: **`imcl install com.ibm.websphere.liberty.v85 -repositories`** *`liberty_repository_dir`* **`-installationDirectory`** *`liberty_install_dir`* **`-acceptLicense`** This command installs Liberty in the *`liberty_install_dir`* directory. The `-acceptLicense` option means that you accept the license terms for the product.
6. Move the directory that contains the servers in a location that does not need specific privileges.

   For the scope of this tutorial, if *`liberty_install_dir`* points to a location where non-administrator or non-root users cannot modify the files, move the directory that contains the servers to a location that does not need specific privileges. In this way, the installation operations can be done without specific privileges.
   a. Go to the installation directory of Liberty.

   b. Create a directory named `etc`. You need administrator or root privileges.
   c. In `etc` directory, create a `server.env` file with the following content:

      ```
      WLP_USER_DIR=<path to a directory where any user can write>
      ```

      For example, on Windows:

      ```
      WLP_USER_DIR=C:\LibertyServers\usr
      ```

7. Create a Liberty server that will be used to install the first node of MobileFirst Server at the later part of the tutorial.
   a. Start a command line.
   b. Go to *liberty_install_dir*/bin, and enter **server create mfp1**.

      This command creates a Liberty server instance named mfp1. You can see its definition at *liberty_install_dir*/usr/servers/mfp1 or *WLP_USER_DIR*/servers/mfp1 (if you modify the directory as described in step 6 on page 6-24).

Results

After the server is created, you can start this server with **server start mfp1** from *liberty_install_dir*/bin/.

To stop the server, enter the command: **server stop mfp1** from *liberty_install_dir*/bin/.

The default home page can be viewed at `http://localhost:9080`.

Note: For production, you need to make sure that the Liberty server is started as a service when the host computer starts. Making the Liberty server start as a service is not part of this tutorial.

Installing MobileFirst Server:
Before you begin

Make sure that Installation Manager V1.8.4 or later is installed. The installation of MobileFirst Server might not succeed with an older version of Installation Manager because the postinstallation operations require Java 7. The older versions of Installation Manager come with Java 6.

About this task

Run Installation Manager to install the binary files of MobileFirst Server on your disk before you create the databases and deploy MobileFirst Server to Liberty profile. In this tutorial, you install MobileFirst Server without IBM MobileFirst Platform Application Center. Application Center is a different component of the product and it is not required to be installed with MobileFirst Server. You need to specify two properties in the command so that Application Center in not installed together with MobileFirst Server. For more information about Application Center, see “Installing and configuring the Application Center” on page 6-198.

You also need to specify one property to indicate whether to activate token licensing or not. In this tutorial, it is assumed that token licensing is not needed and the steps to configure MobileFirst Server for token licensing are not included. However, for production installation, you must determine whether you need to activate token licensing or not. If you do not have a contract to use token licensing with the Rational License Key Server, you do not need to activate token licensing.

If you activate token licensing, you must configure MobileFirst Server for token licensing. For more information, see "Installing and configuring for token licensing" on page 6-150.

In this tutorial, you specify the properties as the parameters through the **imcl** command line. This specification can also be done by using a response file.

Procedure

1. Review the license agreement for MobileFirst Server. The license files can be viewed when you download the installation repository from Passport Advantage.
2. Extract the compressed file of MobileFirst Server installer, that you downloaded, to a folder.

   In the steps that follow, the directory where you extract the installer is referred as `mfp_repository_dir`. It contains a `MobileFirst_Platform_Server/disk1` folder.
3. Start a command line and go to *installation_manager_install_dir*`/tools/eclipse/`.
4. If you accept the license agreement after the review in step 1, install MobileFirst Server.

   Enter the command: **imcl install com.ibm.mobilefirst.foundation.server -repositories** *mfp_repository_dir*/MobileFirst_Platform_Server/disk1 **-properties user.appserver.selection2=none,user.database.selection2=none,user.database.preinstalled=false,user.licensed.by.tokens=false,user.use.ios.edition=false -acceptLicense**

   The following properties are defined to have an installation without Application Center:
   - **user.appserver.selection2**=none
   - **user.database.selection2**=none
   - **user.database.preinstalled**=false

   This property indicates that token licensing is not activated: **user.licensed.by.tokens**=false.

   Set the value of the **user.use.ios.edition** property to `false` to install IBM MobileFirst Platform Foundation.

Results

An installation directory that contains the resources to install MobileFirst components is installed.

You can find the resources in the following folders:
- `MobileFirstServer` folder for MobileFirst Server
- `PushService` folder for MobileFirst Server push service
- `ApplicationCenter` folder for Application Center
- `Analytics` folder for MobileFirst Analytics

The goal of this tutorial is to install MobileFirst Server by using the resources in `MobileFirstServer` folder.

You can also find some shortcuts for the Server Configuration Tool, Ant, and **mfpadm** program in the `shortcuts` folder.

Creating a database:
About this task

This task is to ensure that a database exists in your DBMS, and that a user is allowed to use the database, create tables in it, and use the tables. You can skip this task if you plan to use Derby database.

The database is used to store the technical data that is used by the various MobileFirst components:
- MobileFirst Server administration service
- MobileFirst Server live update service
- MobileFirst Server push service
- MobileFirst runtime

In this tutorial, the tables for all the components are placed under the same schema.

Note: The steps in this task are for DB2. If you plan to use MySQL or Oracle, see "Database requirements" on page 6-65.

Procedure
1. Log on to the computer that is running the DB2 server. It is assumed that a DB2 user, for example named as **`mfpuser`**, exists.
2. Verify that this DB2 user has the access to a database with a page size 32768 or more, and is allowed to create implicit schemas and tables in that database.

   By default, this user is a user declared on the operating system of the computer that runs DB2. That is, a user with a login for that computer. If such user exists, the next action in step 3 is not needed.
3. Create a database with the correct page size for this installation if you do not have one.
   a. Open a session with a user that has `SYSADM` or `SYSCTRL` permissions. For example, use the user **`db2inst1`** that is the default admin user that is created by the DB2 installer.
   b. Open a DB2 command line processor:
      - On Windows systems, click **Start** > **IBM DB2** > **Command Line Processor**.
      - On Linux or UNIX systems, go to `~/sqllib/bin` (or *`db2_install_dir`*`/bin` if `sqllib` is not created in the administrator's home directory) and enter `./db2`.

   c. Enter the following SQL statements to create a database that is called **`MFPDATA`**:
```
CREATE DATABASE MFPDATA COLLATE USING SYSTEM PAGESIZE 32768
CONNECT TO MFPDATA
GRANT CONNECT ON DATABASE TO USER mfpuser
GRANT CREATETAB ON DATABASE TO USER mfpuser
GRANT IMPLICIT_SCHEMA ON DATABASE TO USER mfpuser
DISCONNECT MFPDATA
QUIT
```
   If you defined a different user name, replace **`mfpuser`** with your own user name.

   Note: The statement does not remove the default privileges granted to PUBLIC in a default DB2 database. For production, you might need to reduce the privileges in that database to the minimum requirement for the product. For more information about DB2 security and an example of the security practices, see DB2 security, Part 8: Twelve DB2 security best practices.

Deploying MobileFirst Server to Liberty with Ant tasks:
About this task

You use the Ant tasks to run the following operations:
- Create the tables in the database that are needed by the MobileFirst applications
- Deploy the web applications of MobileFirst Server (the runtime, administration service, live update service, push service components, and MobileFirst Operations Console) to Liberty server.

The following MobileFirst applications are not deployed by Ant tasks:

MobileFirst Analytics
MobileFirst Analytics is typically deployed on a different set of servers than MobileFirst Server because of its high memory requirements. MobileFirst Analytics can be installed manually or with Ant tasks. If it is already installed, you can enter its URL, the user name, and password to send data to it in the Server Configuration Tool. The Server Configuration Tool then configures the MobileFirst apps to send data to MobileFirst Analytics. For more information about the installation of MobileFirst Analytics, see "MobileFirst Analytics Server installation guide" on page 11-2.

Application Center
This application can be used to distribute mobile apps internally to the employees that use the apps, or for test purpose. It is independent of MobileFirst Server and is not necessary to install together with MobileFirst Server. For more information, see "Installing and configuring the Application Center" on page 6-198.

Procedure

Pick the appropriate XML file that contains the Ant tasks and configure the properties.
1. Make a copy of the `*mfp_install_dir*/MobileFirstServer/configuration-samples/configure-liberty-db2.xml` file to a working directory.

   This file contains the Ant tasks for installing MobileFirst Server on Liberty with DB2 as the database. Before you use it, define the properties to describe where the applications of MobileFirst Server are to be deployed.
2. Edit the copy of the XML file and set the values of the following properties:
   - **`mfp.admin.contextroot`** to `/mfpadmin`
   - **`mfp.runtime.contextroot`** to `/mfp`
   - **`database.db2.host`** to the value to the host name of the computer that runs your DB2 database. If the database is on the same computer as Liberty, use `localhost`.
   - **`database.db2.port`** to the port to which the DB2 instance is listening. By default, it is 50000.
   - **`database.db2.driver.dir`** to the directory that contains your DB2 driver: `db2jcc4.jar` and `db2jcc_license_cu.jar`. In a standard DB2 distribution, these files are found in *`db2_install_dir`*`/java`.

- **database.db2.mfp.dbname** to MFPDATA - the database name that you create in "Creating a database" on page 6-27.
- **database.db2.mfp.schema** to MFPDATA - the value of the schema where the tables for MobileFirst Server are to be created. If your DB user is not able to create a schema, set the value to an empty string. For example, **database.db2.mfp.schema**="".
- **database.db2.mfp.username** to the DB2 user that creates the tables. This user also uses the tables at run time. For this tutorial, use mfpuser.
- **appserver.was.installdir** to the Liberty installation directory.
- **appserver.was85liberty.serverInstance** to mfp1 - the value to the name of the Liberty server where MobileFirst Server is to be installed.
- **mfp.farm.configure** to false to install MobileFirst Server in stand-alone mode.
- **mfp.analytics.configure** to false. The connection to MobileFirst Analytics is not in the scope of this tutorial. You can ignore the other properties **mfp.analytics.****.
- **mfp.admin.client.id** to admin-client-id.
- **mfp.admin.client.secret** to adminSecret (or choose another secret password).
- **mfp.push.client.id** to push-client-id.
- **mfp.push.client.secret** to pushSecret (or choose another secret password).
- **mfp.config.admin.user** to the user name of the MobileFirst Server live update service. In a server farm topology, the user name must be the same for all the members of the farm.
- **mfp.config.admin.password** to the password of the MobileFirst Server live update service. In a server farm topology, the password must be the same for all the members of the farm.

3. Keep the default values of the following properties as-is:
   - **mfp.admin.console.install** to true
   - **mfp.admin.default.user** to admin - the name of a default user that is created to log in to MobileFirst Operations Console.
   - **mfp.admin.default.user.initialpassword** to admin - the password of a default user that is created to log in to the admin console.
   - **appserver.was.profile** to Liberty. If the value is different, the Ant task assumes that the installation is on a WebSphere Application Server server.
4. Save the file after the properties are defined.
5. Run the command *mfp_server_install_dir*/shortcuts/**ant -f** configure-liberty-db2.xml. This command shows a list of possible targets for the Ant file.
6. Run *mfp_server_install_dir*/shortcuts/**ant -f** configure-liberty-db2.xml **databases** to create the database tables.
7. Run *mfp_server_install_dir*/shortcuts/**ant -f** configure-liberty-db2.xml **install** to install MobileFirst Server.

Note: If you do not have DB2, and want to test the installation with an embedded Derby as a database, use the *mfp_install_dir*/MobileFirstServer/configuration-samples/configure-liberty-derby.xml file. However, you cannot do the last step of this tutorial ("Creating a farm of two Liberty servers that run MobileFirst Server" on page 6-36) because the Derby database cannot be accessed by multiple Liberty servers. You must set the properties except the DB2 related ones (**database.db2...**). For Derby, set the value of the property

database.derby.datadir to the directory where Derby database can be created. Also, set the value of the property **database.derby.mfp.dbname** to MFPDATA.

Results

You can see a detail of the operations done in the log file of the Ant tasks.

The following operations are run by the Ant tasks:

1. The tables for the following components are created in the database:
   - The administration service and the live update service. Created by the admdatabases Ant target.
   - The runtime component. Created by the rtmdatabases Ant target.
   - The push service. Created by the pushdatabases Ant target.
2. The WAR files of the various components are deployed to Liberty server. You can see the details of the operations in the log under adminstall, rtminstall, and pushinstall targets.

If you have access to the DB2 server, you can list the tables that are created by using these instructions:

1. Open a DB2 command line processor with **mfpuser** as described in step 3 on page 6-27 of "Creating a database" on page 6-27.
2. Enter the SQL statements:
   ```
   CONNECT TO MFPDATA USER mfpuser USING mfpuser_password
   LIST TABLES FOR SCHEMA MFPDATA
   DISCONNECT MFPDATA
   QUIT
   ```

Take note of the following database factors:

Database user consideration
In this tutorial, only one database user is needed. This user is used to create the tables, but is also used as the data source user in the application server at run time. In production environment, you might want to restrict the privileges of the user that is used at run time to the strict minimum (SELECT / INSERT / DELETE / UPDATE), and thus provide a different user for deployment in the application server. For more information about the privileges that are required at run time, see "Database users and privileges" on page 6-64. The Ant files that are provided as examples also use the same users for both cases. However, in the case of DB2, you might want to create your own versions of files. As such, you can distinguish the user that is used to create the databases from the user that is used for the data source in the application server with the Ant tasks.

Database tables creation
For production, you might want to create the tables manually. For example, if your DBA wants to override some default settings or assign specific table spaces. The database scripts that are used to create the tables are available in *mfp_server_install_dir*/MobileFirstServer/databases and *mfp_server_install_dir*/PushService/databases. For more information, see "Create the database tables manually" on page 6-67.

The server.xml file and some application server setting are modified during the installation. Before each modification, a copy of the server.xml file is made, such as server.xml.bak, server.xml.bak1, and server.xml.bak2. To see everything that was added, you can compare the server.xml file with the oldest backup (server.xml.bak). On Linux, you can use the command **diff --strip-trailing-cr**

server.xml server.xml.bak to see the differences. On AIX, use the command **diff server.xml server.xml.bak** to find the differences.

Modification of the application server settings (specific to Liberty):

1. The Liberty features are added.

   The features are added for each application and can be duplicated. For example, the JDBC feature is used for both the administration service and the runtime components. This duplication allows the removal of the features of an application when it is uninstalled without breaking the other applications. For example, if you decide at some point to uninstall the push service from a server and install it on another server. However, not all topologies are possible. The administration service, the live update service, and the runtime component must be on the same application server with Liberty profile. For more information, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84. The duplication of features does not create issue unless the features that added are conflicting. Adding the `jdbc-40` and `jdbc-41` features would cause a problem, but adding twice the same feature does not.
2. `host='*'` is added in the `httpEndPoint` declaration.

   This setting is to allow the connection to the server from all network interfaces. In production, you might want to restrict the host value of the HTTP endpoint.
3. The `tcpOptions` element (`tcpOptions soReuseAddr="true"`) is added in the server configuration to enable immediate rebind to a port with no active listener and improve the throughput of the server.
4. A keystore with ID **defaultKeyStore** is created if it does not exist.

   The keystore is to enable the HTTPS port and more specifically, to enable the JMX communication between the administration service (`mfp-admin-service.war`) and the runtime component (`mfp-server.war`). The two applications communicate via JMX. In the case of Liberty profile, `restConnector` is used to communicate between the applications in a single server and also between the servers of a Liberty Farm. It requires the use of HTTPS. For the keystore that is created by default, Liberty profiles creates a certificate with a validity period of 365 days. This configuration is not intended for production use. For production, you need to reconsider to use your own certificate.

   To enable JMX, a user with administrator role (named **MfpRESTUser**) is created in the basic registry. Its name and password are provided as JNDI properties (**mfp.admin.jmx.user** and **mfp.admin.jmx.pwd**) and are used by the runtime component and the administration service to run JMX queries. In the global JMX properties, some properties are used to define the cluster mode (stand-alone server or working in a farm). The Ant tasks set the **mfp.topology.clustermode** property to `Standalone` in Liberty server. In the later part of this tutorial about the creation of a farm, the property is modified to `Cluster`.
5. The creation of users (Also valid for Apache Tomcat and WebSphere Application Server)
   - Optional users: The default Ant task creates a test user (`admin/admin`) so that you can use this user to log to the console after the installation.
   - Mandatory users: Except for the server farm topology, the Ant task also creates a user (named as **configUser_mfpadmin** with a randomly generated password) to be used by the administration service to

contact the local live update service. For the server farm topology, this user and the password must be the same for all the members of the farm. For Liberty server, **MfpRESTUser** is created. If your application server is not configured to use a basic registry (but for example, an LDAP registry), the Ant task attempts to create a user in the basic registry. This operation fails and might create an invalid `server.xml`. In this case, you need to modify the Ant files so that users are not created, as documented in "Installing with Ant Tasks" on page 6-110.

6. The `webContainer` element is modified.

   The `deferServletLoad` web container custom property is set to false. Both the runtime component and the administration service must start when the server starts. These components can thus register the JMX beans and start the synchronization procedure that allows the runtime component to download all the applications and adapters that it needs to serve.

7. The default executor is customized to set large values to **coreThreads** and **maxThreads**.

   This setting avoids timeout issues that break the startup sequence of the runtime component and administration service on some Liberty versions. The absence of this statement can be the cause of these errors in the server log file:

```
Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: Read timed out
FWLSE3000E: A server error was detected.
FWLSE3012E: JMX configuration error. Unable to obtain MBeans. Reason: "Read timed out".
```

Declaration of applications

The following applications are installed:

- `mfpadmin`, the administration service
- `mfpadminconfig`, the live update service
- `mfpconsole`, MobileFirst Operations Console
- `mobilefirst`, MobileFirst runtime component
- `imfpush`, the push service

You can separate the applications in different application servers, but under certain constraints that are documented in "Topologies and network flows" on page 6-78. You can either use Ant tasks or install the product manually.

Administration service

The administration service is the service for managing MobileFirst applications, adapters, and their configurations. It is secured by security roles. By default, the Server Configuration Tool adds a user (**admin**) with the administrator role, that you can use to log in to the console for testing. The configuration of the security role must be done after an installation with the Server Configuration Tool (or with Ant tasks). See "Configuring user authentication for MobileFirst Server administration" on page 6-166. You might want to map the users or the groups that come from the basic registry or an LDAP registry that you configure in your application server to each security role.

The class loader delegation is set to parent last for Liberty profile and WebSphere Application Server, and for all MobileFirst applications. This setting is to avoid conflicts between the classes packaged in the MobileFirst applications and the classes of the application server. Forgetting to set the class loader delegation to parent last is a frequent source of error in manual installation. For Apache Tomcat, this declaration is not needed.

In Liberty profile, a common library is added to the application for decrypting passwords that are passed as JNDI properties. The Server Configuration Tool defines two mandatory JNDI properties for the administration service: **`mfp.config.service.user`** and **`mfp.config.service.password`**. They are used by the administration service to connect to the live update service with its REST API. More JNDI properties can be defined to tune the application or adapt it to your installation particularities. For more information, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

The data source to the database that contains the tables for the administration service is declared, as well as a library for its JDBC driver.

Live update service

The live update service stores information about the runtime and application configurations. It is controlled by the administration service and must always run on the same server as the administration service. The context root is *`context_root_of_admin_server`*`config`. As such, it is `mfpadminconfig`. The administration service assumes that this convention is respected to create the URL of its requests to the REST services of the live update service.

The class loader is set with delegation parent last as discussed in the administration service section.

The live update service has one security role, **`admin_config`**. A user must be mapped to that role. Its password and login must be provided to the administration service with the JNDI property: **`mfp.config.service.user`** and **`mfp.config.service.password`**. In a server farm topology, the user and its password must be the same for all the members of the farm.

For information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174 and "List of JNDI properties for MobileFirst Server live update service" on page 6-182.

It also needs a data source with JNDI name on Liberty profile. The convention is *`context_root_of_config_server`*`/jdbc/ConfigDS`. In this tutorial, it is defined as `mfpadminconfig/jdbc/ConfigDS`. In an installation by the Server Configuration Tool or with Ant tasks, the tables of the live update service are in the same database and schema as the tables of the administration service. The user to access these tables is also the same.

MobileFirst Operations Console

MobileFirst Operations Console is declared with the same security roles as the administration service. The users that are mapped to the security roles of MobileFirst Operations Console must also be mapped to the same security role of the administration service.

Indeed, MobileFirst Operations Console runs queries to the administration service on the behalf of the console user.

The Server Configuration Tool positions one JNDI property, **mfp.admin.endpoint**, that indicates how the console connects to the administration service. The default value set by the Server Configuration Tool is `'*://*:*/mfpadmin'`. The setting means that it must use the same protocol, host name, and port as the incoming HTTP request to the console, and the context root of the administration service is `/mfpadmin`. If you want to force the request to go though a web proxy, change the default value. For more information about the possible values for this URL, or for information about other possible JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

The class loader is set with delegation parent last as discussed in the administration service section.

MobileFirst runtime

This application is not secured by a security role. It is not required to log in with a user known by the Liberty server to access this application. The mobile devices requests are routed to the runtime. They are authenticated by other mechanism specific to the product (such as OAuth) and the configuration of the MobileFirst applications.

The class loader is set with delegation parent last as discussed in the administration service section.

It also needs a data source with JNDI name on Liberty profile. The convention is *`context_root_of_runtime`*`/jdbc/mfpDS`. In this tutorial, it is defined as `mobilefirst/jdbc/mfpDS`. In an installation by the Server Configuration Tool or with Ant tasks, the tables of the runtime are in the same database and schema as the tables of the administration service. The user to access these tables is also the same.

Push service

This application is secured by OAuth. The valid OAuth tokens must be included in any HTTP request to the service.

The configuration of OAuth is made through the JNDI properties (such as the URL of the authorization server, the client ID, and the password of the push service). The JNDI properties also indicate the security plug-in (**mfp.push.services.ext.security**) and the fact that a relational database is used (**mfp.push.db.type**). The requests from the mobile devices to the push service are routed to this service. The context root of the push service must be `/imfpush`. The client SDK computes the URL of the push service based on the URL of the runtime with the context root (`/imfpush`). If you want to install the push service on a different server than the runtime, you need to have an HTTP router that can route the device requests to the relevant application server.

The class loader is set with delegation parent last as discussed in the administration service section.

It also needs a data source with JNDI name on Liberty profile. The JNDI name is `imfpush/jdbc/imfPushDS`. In an installation by the Server Configuration Tool or with Ant tasks, the tables of the push service are in the same database and schema as the tables of the administration service. The user to access these tables is also the same.

Other files modification

The Liberty profile `jvm.options` file is modified. A property **`com.ibm.ws.jmx.connector.client.rest.readTimeout`** is defined to avoid timeout issues with JMX when the runtime synchronizes with the administration service.

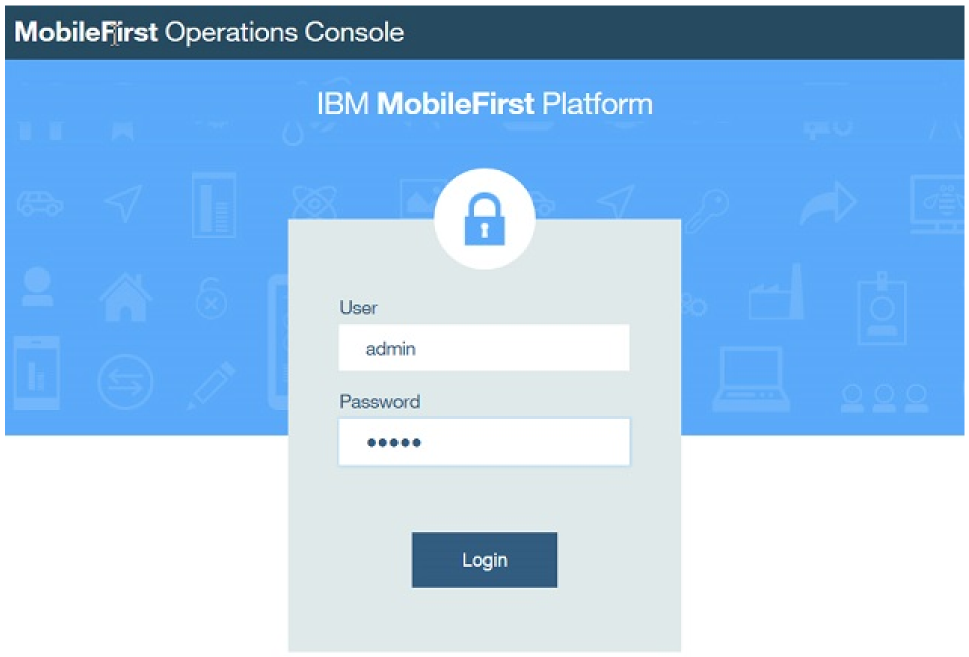
Testing the installation:
About this task

After the installation is complete, you can use this procedure to test the components that are installed.

Procedure

1. Start the server by using the command **`server start mfp1`**. The binary file for the server is in *`liberty_install_dir`*`/bin`.
2. Test MobileFirst Operations Console with a web browser.

   Go to `http://localhost:9080/mfpconsole`. By default, the server runs on port 9080. However, you can verify the port in the element `<httpEndpoint>` as defined in the `server.xml` file. A login screen is displayed.

3. Log in with `admin/admin`.

   This user is created by default.

   Note: If you connect with HTTP, the login ID and password are sent in clear text in the network. For a secure login, use HTTPS to log to the server. You can see the HTTPS port of the Liberty server in the **`httpsPort`** attribute of the `<httpEndpoint>` element in the `server.xml` file. By default, the value is 9443.

4. Log out of the console with **Hello Admin** > **Sign Out**.
5. Enter the following URL: `https://localhost:9443/mfpconsole` in the web browser and accept the certificate.

   By default, the Liberty server generates a default certificate that is not known by your web browser, you need to accept the certificate. Mozilla Firefox presents this certification as a security exception.
6. Log in again with `admin/admin`.

   The login and password are encrypted between your web browser and MobileFirst Server. In production, you might want to close the HTTP port.

Creating a farm of two Liberty servers that run MobileFirst Server:
About this task

In this task, you will create a second Liberty server that runs the same MobileFirst Server and connected to the same database. In production, you might use more than one server for performance reasons, to have enough servers to serve the number of transactions per second that is needed for your mobile applications at peak time. It is also for high availability reasons to avoid having a single point of failure.

When you have more than one server that runs MobileFirst Server, the servers must be configured as a farm. This configuration enables any administration service to contact all the runtimes of a farm. If the cluster is not configured as a farm, only the runtime that runs in the same application server as the management service that runs the management operation is notified. Others runtimes are not aware of the change. For example, you deploy a new version of an adapter in a cluster that is not configured as a farm, only one server would serve the new adapter. The other servers would continue to serve the old adapter. The only situation where you can have a cluster and do not need to configure a farm is when you install your servers on WebSphere Application Server Network Deployment. The administration service is able to find all the servers by querying the JMX beans with the deployment manager. The deployment manager must be running to allow management operations because it is used to provide the list of the MobileFirst JMX beans of the cell.

When you create a farm, you also need to configure an HTTP server to send queries to all the members of the farm. The configuration of an HTTP server is not included in this tutorial. This tutorial is only about configuring the farm so that management operations are replicated to all the runtime components of the cluster.

Procedure

1. Create a second Liberty server on the same computer.
   a. Start a command line.
   b. Go to *`liberty_install_dir`*`/bin`, and enter **`server create mfp2`**.
2. Modify the HTTP and HTTPS ports of the server mfp2 so that they do not conflict with the ports of server mfp1.
   a. Go to the second server directory.

      The directory is *`liberty_install_dir`*`/usr/servers/mfp2` or *`WLP_USER_DIR`*`/servers/mfp2` (if you modify the directory as described in step 6 on page 6-24 of "Installing WebSphere Application Server Liberty Core" on page 6-24).
   b. Edit the `server.xml` file. Replace

```
<httpEndpoint id="defaultHttpEndpoint"
    httpPort="9080"
    httpsPort="9443" />
```

with

```
<httpEndpoint id="defaultHttpEndpoint"
    httpPort="9081"
    httpsPort="9444" />
```

The HTTP and HTTPS ports of the server mfp2 do not conflict with the ports of the server mfp1 with this change. Make sure to modify the ports before you run the installation of MobileFirst Server. Otherwise, if you modify the port after the installation is made, you also need to reflect the change of the port in the JNDI property: **`mfp.admin.jmx.port`**.

3. Copy the Ant file that you used in "Deploying MobileFirst Server to Liberty with Ant tasks" on page 6-28, and change the value of the property **`appserver.was85liberty.serverInstance`** to `mfp2`. The Ant tasks detect that the databases exist and do not create the tables (see the following log extract). Then, the applications are deployed to the server.

```
[configuredatabase] Checking connectivity to MobileFirstAdmin database MFPDATA with schema 'MFPDATA' and user 'mfpuser'...
[configuredatabase] Database MFPDATA exists.
[configuredatabase] Connection to MobileFirstAdmin database MFPDATA with schema 'MFPDATA' and user 'mfpuser' succeeded.
[configuredatabase] Getting the version of MobileFirstAdmin database MFPDATA...
[configuredatabase] Table MFPADMIN_VERSION exists, checking its value...
[configuredatabase] GetSQLQueryResult => MFPADMIN_VERSION = 8.0.0
[configuredatabase] Configuring MobileFirstAdmin database MFPDATA...
[configuredatabase] The database is in latest version (8.0.0), no upgrade required.
[configuredatabase] Configuration of MobileFirstAdmin database MFPDATA succeeded.
```

4. Test the two servers with HTTP connection.
   a. Open a web browser.
   b. Enter the following URL: `http://localhost:9080/mfpconsole`. The console is served by server mfp1.
   c. Log in with `admin/admin`.
   d. Open a tab in the same web browser and enter the URL: `http://localhost:9081/mfpconsole`. The console is served by server mfp2.
   e. Log in with `admin/admin`. If the installation is done correctly, you can see the same welcome page in both tabs after login.
   f. Return to first browser tab and click **Hello, admin** > **Download Audit Log**. You are logged out of the console and see the login screen again.

      This logout behavior is an issue. The problem happens because when you log on to server mfp2, a Lightweight Third Party Authentication (LTPA) token is created and stored in your browser as a cookie. However, this LTPA token is not recognized by server mfp1. Switching between servers is likely to happen in a production environment when you have an HTTP load balancer in front of the cluster. To resolve this issue, you must ensure that both servers (mfp1 and mfp2) generate the LTPA tokens with the same secret keys. Copy the LTPA keys from server mfp1 to server mfp2.

      1) Stop both servers with these commands:

      ```
      server stop mfp1
      server stop mfp2
      ```

      2) Copy the LTPA keys of server mfp1 to server mfp2.

         From *`liberty_install_dir`*`/usr/servers` or *`WLP_USER_DIR`*`/servers`, run the following command depending on your operating system.
         - On UNIX: **`cp mfp1/resources/security/ltpa.keys mfp2/resources/security/ltpa.keys`**

- On Windows: **copy mfp1/resources/security/ltpa.keys mfp2/resources/security/ltpa.keys**

g. Restart the servers. Switch from one browser tab to another other does not require you to relogin. In a Liberty server farm, all servers must have the same LTPA keys.

5. Enable the JMX communication between the Liberty servers.

   The JMX communication with Liberty, is done via the Liberty REST connector over the HTTPS protocol. To enable this communication, each server of the farm must be able to recognize the SSL certificate of the other members. You need to exchange the HTTPS certificates in their truststores. Use IBM utilities such as Keytool, which is part of the IBM JRE distribution in `java/bin` to configure the truststore. 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 *`WLP_USER_DIR`*`/servers/`*`server_name`*`/resources/security/key.jks`. The password of this default keystore, as can be seen in the `server.xml` file, is `mobilefirst`.

   Tip: You can change it with the Keytool utility, but you must also change the password in the `server.xml` file so that Liberty server can read that keystore. In this tutorial, use the default password.

   a. In *`WLP_USER_DIR`*`/servers/mfp1/resources/security`, enter **`keytool -list -keystore key.jks`**. The command shows the certificates in the keystore. There is only one named **`default`**. You are prompted for the password of the keystore (`mobilefirst`) before you can see the keys. This is the case for all the next commands with Keytool utility.

   b. Export the default certificate of server mfp1 with the command: **`keytool -exportcert -keystore key.jks -alias default -file mfp1.cert`**.

   c. In *`WLP_USER_DIR`*`/servers/mfp2/resources/security`, export the default certificate of server mfp2 with the command: **`keytool -exportcert -keystore key.jks -alias default -file mfp2.cert`**.

   d. In the same directory, import the certificate of server mfp1 with the command: **`keytool -import -file ../../../mfp1/resources/security/mfp1.cert -keystore key.jks`** The certificate of server mfp1 is imported into the keystore of server mfp2 so that server mfp2 can trust the HTTPS connections to server mfp1. You are asked to confirm that you trust the certificate.

   e. In *`WLP_USER_DIR`*`/servers/mfp1/resources/security`, import the certificate of server mfp2 with the command: **`keytool -import -file ../../../mfp2/resources/security/mfp2.cert -keystore key.jks`**. After this step, the HTTPS connections between the two servers are possible.

6. Modify the JNDI properties of each server to configure the farm.

   a. Edit the *`WLP_USER_DIR`*`/servers/mfp1/server.xml` file.

      1) Set the value of **`mfp.topology.clustermode`** property to `Farm`. The administrations service operates in farm mode with this setting.

      2) Add this JNDI entry: `<jndiEntry jndiName="mfp.admin.serverid" value="mfp1"/>`. Each server of the farm needs to have a unique server ID.

      3) Review the value of **`mfp.admin.jmx.host`** property. In this tutorial, the value is set to `localhost`. This setting is acceptable if all clusters of the farm are running on the same computer. However, in general, the host name must be resolvable by all the members of the farm. Set the host name of the computer as the value of **`mfp.admin.jmx.host`**.

a. Edit the *WLP_USER_DIR*/servers/mfp2/server.xml file and proceed with the same changes. For this JNDI entry (**mfp.admin.serverid**), you must give a value that is different from mfp1. Use mfp2.

Note: This procedure shows you the complete manual steps to configure the already installed Liberty server as the members of a farm. If you plan to install the server farm members, some steps can be automated by Ant tasks. You can configure the Ant tasks by setting the values of the following properties:

- Set **mfp.farm.configure** to true.
- **mfp.farm.server.id**: An identifier that you define for each farm member. This identifier must be unique across all farm members.

For more information, see "Installing a server farm with Ant tasks" on page 6-141.

Testing the farm and see the changes in MobileFirst Operations Console:
Procedure

1. Start the two servers:

```
server start mfp1
server start mfp2
```

2. Access the console. For example, http://localhost:9080/mfpconsole, or https://localhost:9443/mfpconsole in HTTPS. In the left sidebar, an extra menu that is labeled as **Server Farm Nodes** appears. If you click **Server Farm Nodes**, you can view the status of each node. You might need to wait a bit for both nodes to be started.

Installing MobileFirst Server for a production environment

This section is intended for developers and administrators who want to install and configure MobileFirst Server for a production environment.

This section provides details, beyond the tutorial about MobileFirst Server installation, to assist you in planning and preparing an installation for your specific environment.

For more information about the configuration of the MobileFirst Server, see "Configuring MobileFirst Server" on page 6-164.

Installation prerequisites

For smooth installation of MobileFirst Server, ensure that you fulfill all the software prerequisites.

Before you install MobileFirst Server, you need to have the following software:

Database Management System (DBMS)

A DBMS is needed to store the technical data of MobileFirst Server components. You must use one of the supported DBMS:

- IBM DB2
- MySQL
- Oracle

For more information about the versions of DBMS that are supported by the product, see "System requirements" on page 2-7. If you use a relational DBMS (IBM DB2, Oracle, or MySQL), you need the JDBC driver for that database during the installation process. The JDBC drivers are not provided by MobileFirst Server installer. Make sure that you have the JDBC driver.

- For DB2, use the DB2 JDBC driver V4.0 (`db2jcc4.jar`).
- For MySQL, use the Connector/J JDBC driver.
- For Oracle, use the Oracle thin JDBC driver.

Java application server

A Java application server is needed to run the MobileFirst Server applications. You can use any of the following application servers:

- WebSphere Application Server Liberty Core
- WebSphere Application Server Liberty Network Deployment
- WebSphere Application Server
- Apache Tomcat

For more information about the versions of application servers that are supported by the product, see "System requirements" on page 2-7. The application server must run with Java 7 or later. By default, some versions of WebSphere Application Server run with Java 6. With this default, they cannot run MobileFirst Server.

IBM Installation Manager V1.8.4 or later

Installation Manager is used to run the installer of MobileFirst Server. You must install Installation Manager V1.8.4 or later. The older versions of Installation Manager are not able to install IBM MobileFirst Platform Foundation V8.0 because the postinstallation operations of the product require Java 7. The older versions of Installation Manager come with Java 6.

Download the installer of IBM Installation Manager V1.8.4 or later from Installation Manager and Packaging Utility download links.

Installation Manager repository for MobileFirst Server

You can download the repository from the IBM MobileFirst Platform Foundation eAssembly on IBM Passport Advantage. The name of the pack is `IBM MobileFirst Platform Foundation V8.0 .zip file of Installation Manager Repository for IBM MobileFirst Platform Server`.

You might also want to apply the latest fix pack that can be downloaded from IBM Support Portal. The fix pack cannot be installed without the repository of the base version in the repositories of Installation Manager.

The IBM MobileFirst Platform Foundation eAssembly includes the following installers:

- IBM DB2 Workgroup Server Edition
- IBM WebSphere Application Server Liberty Core

For Liberty, you can also use IBM WebSphere SDK Java Technology edition with IBM WebSphere Application Server Liberty Core supplement.

Running IBM Installation Manager

IBM Installation Manager installs the IBM MobileFirst Platform Server files and tools on your computer.

You run Installation Manager to install the binary files of MobileFirst Server and the tools to deploy the MobileFirst Server applications to an application server on your computer. The files and tools that are installed by the installer are described in "Distribution structure of MobileFirst Server" on page 6-61.

You need IBM Installation Manager V1.8.4 or later to run the MobileFirst Server installer. You can run it either in graphical mode or in command line mode.

Two main options are proposed during the installation process:
- Activation of token licensing
- Installation and deployment of IBM MobileFirst Platform Application Center

Token licensing

Token licensing is one of the two licensing methods supported by MobileFirst Server. You must determine whether you need to activate token licensing or not. If you do not have a contract that defines the use of token licensing with the Rational License Key Server, do not activate token licensing. If you activate token licensing, you must configure MobileFirst Server for token licensing. For more information, see "Installing and configuring for token licensing" on page 6-150.

IBM MobileFirst Platform Application Center

Application Center is a component of IBM MobileFirst Platform Foundation. With Application Center, you can share mobile applications that are under development within your organization in a single repository of mobile applications.

If you choose to install Application Center with Installation Manager, you must provide the database and the application server parameters so that Installation Manager configures the databases and deploys Application Center to the application server. If you choose not to install Application Center with Installation Manager, Installation Manager saves the WAR file and the resources of Application Center to your disk. It does not set up the databases nor deploys Application Center WAR file to your application server. You can do this later by using Ant tasks or manually. This option to install Application Center is a convenient way to discover Application Center because you are guided during the installation process by the graphical Install wizard.

However, for production installation, use Ant tasks to install Application Center. The installation with Ant tasks enables you to decouple the updates to MobileFirst Server from the updates to Application Center.

- Advantage of installing Application Center with Installation Manager.
  - A guided graphical wizard assists you through the installation and deployment process.
- Disadvantages of installing Application Center with Installation Manager.
  - If Installation Manager is run with the root user on UNIX or Linux, it might create files that are owned by root in the directory of the application server where Application Center is deployed. As a result, you must run the application server as root.
  - You have no access to the database scripts and cannot provide them to your database administrator to create the tables before you run the installation procedure. Installation Manager creates the database tables for you with default settings.

- Each time when you upgrade the product, for example to install an interim fix, Application Center is upgraded first. The upgrade of Application Center includes operations on the database and the application server. If the upgrade of Application Center fails, it prevents Installation Manager from completing the upgrade, and prevents you from upgrading other MobileFirst Server components. For production installation, do not deploy Application Center with Installation Manager. Install Application Center separately with Ant tasks after Installation Manager installsMobileFirst Server. For more information about Application Center, see "Installing and configuring the Application Center" on page 6-198.

Important: The MobileFirst Server installer installs only the MobileFirst Server binary files and tools on your disk. It does not deploy the MobileFirst Server applications to your application server. After you run the installation with Installation Manager, you must set up the databases and deploy the MobileFirst Server applications to your application server. For more information, see:
- "Setting up databases" on page 6-63
- "Topologies and network flows" on page 6-78
- "Installing MobileFirst Server to an application server" on page 6-100

Similarly, when you run Installation Manager to update an existing installation, it updates only the files on your disk. You need to perform more actions to update the applications that are deployed to your application servers. To apply an interim fix or fix pack, see:
- Applying a fix pack or an interim fix with the Server Configuration Tool
- "Applying a fix pack by using the Ant files" on page 6-111

Related tasks:
"Installing IBM Installation Manager" on page 6-6

Administrator versus user mode:

You can install MobileFirst Server in two different IBM Installation Manager modes. The mode depends on how IBM Installation Manager itself is installed. The mode determines the directories and commands that you use for both Installation Manager and packages.

IBM MobileFirst Platform Foundation supports the following two Installation Manager modes:
- Administrator mode
- User (nonadministrator) mode

Group mode that is available on Linux or UNIX is not supported by the product.

Administrator mode
In administrator mode, Installation Manager must be run as root under Linux or UNIX, and with administrator privileges under Windows. The repository files of Installation Manager (that is the list of installed software and its version) are installed in a system directory. `/var/ibm` on Linux or UNIX, or `ProgramData` on Windows. Do not deploy Application Center with Installation Manager if you run Installation Manager in administrator mode. For more information about not deploying Application Center in administrator mode, see Disadvantages of installing Application Center with Installation Manager in "Running IBM Installation Manager" on page 6-40.

If you install other IBM software (such as WebSphere Application Server or Liberty) with Installation Manager in administrator mode, you might want to move the servers to a directory that can be written by non-root users. For WebSphere Application Server, create a profile in a different location. For Liberty, move the directory `usr` (that contains the server) to a location that can be written by other users. For an example, see step 6 on page 6-7 in "Installing WebSphere Application Server Liberty Core" on page 6-6 of the Getting started tutorial.

User (nonadministrator) mode

In user mode, Installation Manager can be run by any user without specific privileges. However, the repository files of Installation manager are stored in the user's home directory. Only that user is able to upgrade an installation of the product.

If you do not run Installation Manager as root, make sure that you have a user account that is available later when you upgrade the product installation or apply an interim fix.

For more information about the Installation Manager modes, see **Installing as an administrator, nonadministrator, or group** in the IBM Installation Manager documentation.

Installing by using IBM Installation Manager Install wizard:

Use the graphical user interface (GUI) of Installation Manager to install MobileFirst Server.

Before you begin

Before you begin with the installation, make sure that Installation Manager V1.8.4 or later is installed. For more information, see "Installing IBM Installation Manager" on page 6-6.

Download the Installation Manager repository for MobileFirst Server from the IBM MobileFirst Platform Foundation eAssembly on IBM Passport Advantage. The name of the image is `IBM MobileFirst Platform Foundation V8.0 .zip file of Installation Manager Repository for IBM MobileFirst Platform Server`.

Lastly, go through the following sections to learn more about the installation options:
- "Administrator versus user mode" on page 6-42
- Token licensing and Application Center installation details

About this task

Follow the steps in the procedure to install the resources of MobileFirst Server, and the tools (such as the Server Configuration Tool, Ant, and **`mfpadm`** program).

The decisions in the following two panes in the installation wizard are mandatory:
- The **General settings** panel.
- The **Choose configuration** panel to install Application Center.

Procedure

1. Launch Installation Manager.
2. Add the repository of MobileFirst Server in Installation Manager.

   a. Go to **File** > **Preferences** and click **Add Repositories...**.
   b. Browse for the repository file in the directory where the installer is extracted.

      If you decompress the IBM MobileFirst Platform Foundation V8.0 `.zip` file for MobileFirst Server in *`mfp_installer_directory`* folder, the repository file can be found at *`mfp_installer_directory`*`/MobileFirst_Platform_Server/disk1/diskTag.inf`.

      You might also want to apply the latest fix pack that can be downloaded from IBM Support Portal. Make sure to enter the repository for the fix pack. If you decompress the fix pack in *`fixpack_directory`* folder, the repository file is found in *`fixpack_directory`*`/MobileFirst_Platform_Server/disk1/diskTag.inf`.

      Note: You cannot install the fix pack without the repository of the base version in the repositories of Installation Manager. The fix packs are incremental installers and need the repository of the base version to be installed.
   c. Select the file and click **OK**.
   d. Click **OK** to close the **Preferences** panel.
3. After you accept the license terms of the product, click **Next**.
4. Choose the package group to install the product.

   IBM MobileFirst Platform Foundation V8.0 is a replacement for the previous releases that have a different installation name:
   - `Worklight` for V5.0.6
   - `IBM Worklight` for V6.0 to V6.3

   If one of these older versions of the product is installed on your computer, Installation Manager offers you an option `Use an Existing Package Group` at the start of the installation process. This option uninstalls your older version of the product, and reuse your older installation options to upgrade IBM MobileFirst Platform Application Center if it was installed. For information about upgrading an older version of the product, see "Upgrading to IBM MobileFirst Platform Foundation V8.0.0" on page 5-1.

   For a separate installation, select the `Create a New Package group` option so that you can install the new version alongside with the older one.

   If no other version of the product is installed on your computer, choose the `Create a new package group` option to install the product in a new package group.
5. Click **Next**.
6. Decide whether to activate token licensing in the **Activate token licensing** section of the **General settings** panel.

   If you have a contract to use token licensing with Rational License Key Server, select `Activate token licensing with the Rational License Key Server` option. After you activate token licensing, you must do extra steps to configure MobileFirst Server. For more information, see "Installing and configuring for token licensing" on page 6-150.

   Otherwise, select `Do not activate token licensing with the Rational License Key Server` option to proceed.
7. Keep the default option (`No`) as-is in the **Install IBM MobileFirst Platform Foundation for iOS** section of the **General settings** panel.
8. Decide whether to install Application Center in **Choose configuration** panel.

For production installation, use Ant tasks to install Application Center. The installation with Ant tasks enables you to decouple the updates to MobileFirst Server from the updates to Application Center. In this case, select `No` option in the **Choose configuration** panel so that Application Center is not installed.

If you select `Yes`, you need to go through the next panes to enter the details about the database you plan to use and the application server where you plan to deploy Application Center. You also need to have the JDBC driver of your database available.

For more information about installing Application Center, see "Installing and configuring the Application Center" on page 6-198.

9. Click **Next** until you reach the **Thank You** panel. Then, proceed with the installation.

Results

An installation directory that contains the resources to install MobileFirst components is installed.

You can find the resources in the following folders:
- `MobileFirstServer` folder for MobileFirst Server
- `PushService` folder for MobileFirst Server push service
- `ApplicationCenter` folder for Application Center
- `Analytics` folder for MobileFirst Analytics

You can also find some shortcuts for the Server Configuration Tool, Ant, and **`mfpadm`** program in the `shortcuts` folder.

Installing by running IBM Installation Manager in command line:

Run Installation Manager in command line mode to install MobileFirst Server.

Before you begin

Before you begin with the installation, make sure to go through the following sections:
- Token licensing and Application Center installation details
- "Administrator versus user mode" on page 6-42

About this task

To get familiar about installing the product with Installation Manager in command line mode, see "Installing IBM Installation Manager" on page 6-23 and "Installing MobileFirst Server" on page 6-25 in the installation tutorial.

The MobileFirst Server installer requires some parameters during the installation process. You need to pass the value of the parameters in the command line for simple cases. Another way is to define them in the XML response file and run the installation by using the response file.

The installation by command line in this procedure does not require a response file. It does not deploy Application Center. You can later deploy it with Ant tasks. For more information, see "Installing and configuring the Application Center" on page 6-198.

However, if you prefer to install and deploy Application Center with Installation Manager, you need a response file. See "Installing by using XML response files (silent installation)" on page 6-47.

Procedure

1. Review the license agreement for MobileFirst Server. The license files can be viewed when you download the installation repository from Passport Advantage.
2. Extract the compressed file of MobileFirst Server repository, that you downloaded, to a folder.

   You can download the repository from the IBM MobileFirst Platform Foundation eAssembly on IBM Passport Advantage. The name of the pack is `IBM MobileFirst Platform Foundation V8.0 .zip file of Installation Manager Repository for IBM MobileFirst Platform Server`.

   In the steps that follow, the directory where you extract the installer is referred as *`mfp_repository_dir`*. It contains a `MobileFirst_Platform_Server/disk1` folder.
3. Start a command line and go to *`installation_manager_install_dir`*`/tools/eclipse/`.
4. If you accept the license agreement after the review in step 1, install MobileFirst Server.
   - For an installation without token licensing enforcement (if you do not have a contract that defines the use of token licensing), enter the command:

     ```
     imcl install com.ibm.mobilefirst.foundation.server -repositories
     mfp_repository_dir/MobileFirst_Platform_Server/disk1 -properties
     user.appserver.selection2=none,user.database.selection2=none,user.database.
     preinstalled=false,user.licensed.by.tokens=false,user.use.ios.edition=false
     -acceptLicense
     ```
   - For an installation with token licensing enforcement, enter the command:

     ```
     imcl install com.ibm.mobilefirst.foundation.server -repositories
     mfp_repository_dir/MobileFirst_Platform_Server/disk1 -properties
     user.appserver.selection2=none,user.database.selection2=none,user.database.
     preinstalled=false,user.licensed.by.tokens=true,user.use.ios.edition=false
     -acceptLicense
     ```

     The value of **`user.licensed.by.tokens`** property is set to `true`. You must configure MobileFirst Server for token licensing. For more information, see "Installing and configuring for token licensing" on page 6-150.

   The following properties are set to install MobileFirst Server without Application Center:
   - **`user.appserver.selection2`**`=none`
   - **`user.database.selection2`**`=none`
   - **`user.database.preinstalled`**`=false`

   This property indicates whether token licensing is activated or not: **`user.licensed.by.tokens`**`=`*`true/false`*.

   Set the value of the **`user.use.ios.edition`** property to `false` to install IBM MobileFirst Platform Foundation.
5. If you want to install with the latest interim fix, add the interim fix repository in the **`-repositories`** parameter. The **`-repositories`** parameter takes a comma-separated list of repositories.
   a. Add the version of the interim fix by replacing `com.ibm.mobilefirst.foundation.server` with

`com.ibm.mobilefirst.foundation.server_`*`version`*. *version* has the form `8.0.0.0-`*`buildNumber`*. For example, if you install the interim fix 8.0.0.0-IF201601031015, enter the command: **`imcl install`** `com.ibm.mobilefirst.foundation.server_8.0.0.00-201601031015` **`-repositories`**`....`

For more information about the **`imcl`** command, see Installation Manager: Installing packages by using **`imcl`** commands.

Results

An installation directory that contains the resources to install MobileFirst components is installed.

You can find the resources in the following folders:
- `MobileFirstServer` folder for MobileFirst Server
- `PushService` folder for MobileFirst Server push service
- `ApplicationCenter` folder for Application Center
- `Analytics` folder for MobileFirst Analytics

You can also find some shortcuts for the Server Configuration Tool, Ant, and **`mfpadm`** program in the `shortcuts` folder.

Installing by using XML response files (silent installation):

If you want to install IBM MobileFirst Platform Application Center with IBM Installation Manager in command line, you need to provide a large list of arguments. In this case, use the XML response files to provide these arguments.

About this task

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 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.

Silent installation is described in the 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 Installation Manager wizard, see "Command-line (silent installation) parameters" on page 6-50.

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`.

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

Table 6-1. 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`
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 6-2. 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 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 6-3. Parameters available for silent installation.

Key	When necessary	Description	Allowed values
`user.use.ios.edition`	Always	Set the value to `false` if you plan to install IBM MobileFirst Platform Foundation. If you plan to install the product for iOS edition, you must set the value to `true`.	`true` or `false`

Table 6-3. Parameters available for silent installation (continued).

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 the value to `true`. If you do not plan to use the product with Rational License Key Server, set the value to `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 6-150.	`true` or `false`
`user.appserver.selection2`	Always	Type of application server. `was` means preinstalled WebSphere Application Server 8.5.5. `tomcat` means Tomcat 7.0.	

Table 6-3. Parameters available for silent installation (continued).

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

Table 6-3. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
`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.

Table 6-3. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
`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 6-3. 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.

Table 6-3. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
user.database.selection2	Always	Type of database management system used to store the databases.	`derby`, `db2`, `mysql`, `oracle`, `none` The value `none` means that the installer will not install the Application Center. If this value is used, both **user.appserver.selection2** and **user.database.selection2** must take the value `none`.
user.database.preinstalled	Always	`true` means a preinstalled database management system, `false` means Apache Derby to install.	`true`, `false`
user.database.derby.datadir	${user.database.selection2} == derby	The directory in which to create or assume the Derby databases.	An absolute directory name.
user.database.db2.host	${user.database.selection2} == db2	The host name or IP address of the DB2 database server.	
user.database.db2.port	${user.database.selection2} == db2	The port where the DB2 database server listens for JDBC connections. Usually 50000.	A number between 1 and 65535.
user.database.db2.driver	${user.database.selection2} == db2	The absolute file name of `db2jcc4.jar`.	An absolute file name.

Table 6-3. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
`user.database.db2.appcenter.username`	${user.database.selection2} == db2	The user name used to access the DB2 database for Application Center.	Non-empty.
`user.database.db2.appcenter.password`	${user.database.selection2} == db2	The password used to access the DB2 database for Application Center, optionally encrypted in a specific way.	Non-empty password.
`user.database.db2.appcenter.dbname`	${user.database.selection2} == db2	The name of the DB2 database for Application Center.	Non-empty; a valid DB2 database name.
`user.database.oracle.appcenter.servicename.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.

Table 6-3. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
`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 the Oracle thin driver jar file. (`ojdbc6.jar` or `ojdbc7.jar`)	An absolute file name.

Table 6-3. Parameters available for silent installation (continued).

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

Table 6-3. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
`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 6-4. Files and subdirectories in the `Analytics` subdirectory

Item	Description
`analytics.ear` and *`analytics`*`-*.war`	The EAR and WAR files to install IBM MobileFirst Analytics.
`configuration-samples`	Contains the sample Ant files to install MobileFirst Analytics with Ant tasks.

Table 6-5. Files and subdirectories in the `ApplicationCenter` subdirectory

Item	Description
`configuration-samples`	Contains the sample Ant files to install Application Center. The Ant tasks create the database table and deploy WAR files to an application server.

Table 6-5. Files and subdirectories in the `ApplicationCenter` subdirectory (continued)

Item	Description
`console`	Contains the EAR and WAR files to install Application Center. The EAR file is uniquely for IBM PureApplication System.
`databases`	Contains the SQL scripts to be used for the manual creation of tables for Application Center.
`installer`	Contains the resources to create the Application Center client.
`tools`	The tools of Application Center.

Table 6-6. Files and subdirectories in the `MobileFirstServer` subdirectory

Item	Description
`mfp-ant-deployer.jar`	A set of MobileFirst Server Ant tasks.
`mfp``-*.war`	The WAR files of the MobileFirst Server components.
`configuration-samples`	Contains the sample Ant files to install MobileFirst Server components with Ant tasks.
`ConfigurationTool`	Contains the binary files of the Server Configuration Tool. The tool is launched from *`mfp_server_install_dir`*`/shortcuts`.
`databases`	Contains the SQL scripts to be used for the manual creation of tables for MobileFirst Server components (MobileFirst Server administration service, MobileFirst Server configuration service, and MobileFirst runtime).
`external-server-libraries`	Contains the JAR files that are used by different tools (such as the authenticity tool and the OAuth security tool).

Table 6-7. Files and subdirectories in the `PushService` subdirectory

Item	Description
`mfp-push-service.war`	The WAR file to install MobileFirst Server push service.
`databases`	Contains the SQL scripts to be used for the manual creation of tables for MobileFirst Server push service.

Table 6-8. Files and subdirectories in the `License` subdirectory

Item	Description
`Text`	Contains the license for IBM MobileFirst Platform Foundation.

Table 6-9. Files and subdirectories in the MobileFirst Server installation directory

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

Table 6-10. Files and subdirectories in the `tools` subdirectory

Item	Description
`tools/apache-ant-<version>`	A binary installation of Apache Ant that is used by the Server Configuration Tool. It can also be used to run the Ant tasks.

Setting up databases

Set up the database to be used by MobileFirst Server components.

The following MobileFirst Server components need to store technical data into a database:

- MobileFirst Server administration service
- MobileFirst Server live update service
- MobileFirst Server push service
- MobileFirst runtime

Note: If multiple runtime instances are installed with different context root, each instance needs its own set of tables.

The database can be a relational database such as IBM DB2, Oracle, or MySQL.

Relational databases (DB2, Oracle, or MySQL)

Each component needs a set of tables. The tables can be created manually by running the SQL scripts specific to each component (see “Create the database tables manually” on page 6-67), by using Ant Tasks, or the Server Configuration Tool. The table names of each component do not overlap. Thus, it is possible to put all the tables of these components under a single schema.

However, if you decide to install multiple instances of MobileFirst runtime, each with its own context root in the application server, every instance needs its own set of tables. In this case, they need to be in different schemas.

Relational databases:

Understand the users, privileges, and database requirement before you set up the database tables.

Ensure that you have one of the following relational databases:

- IBM DB2
- MySQL
- Oracle

For more information about the versions of database that are supported by the product, see “System requirements” on page 2-7.

Database users and privileges:

At run time, the MobileFirst Server applications in the application server use data sources as resources to obtain connection to relational databases. The data source needs a user with certain privileges to access the database.

You need to configure a data source for each MobileFirst Server application that is deployed to the application server to have the access to the relational database. The data source requires a user with specific privileges to access the database. The number of users that you need to create depends on the installation procedure that is used to deploy MobileFirst Server applications to the application server.

Installation with the Server Configuration Tool
The same user is used for all components (MobileFirst Server administration service, MobileFirst Server configuration service, MobileFirst Server push service, and MobileFirst runtime)

Installation with Ant tasks
The sample Ant files that are provided in the product distribution use the same user for all components. However, it is possible to modify the Ant files to have different users:

- The same user for the administration service and the configuration service as they cannot be installed separately with Ant tasks.
- A different user for the runtime
- A different user for the push service.

Manual installation
It is possible to assign a different data source, and thus a different user, to each of the MobileFirst Server components.

At run time, the users must have the following privileges on the tables and sequences of their data:

- `SELECT TABLE`
- `INSERT TABLE`
- `UPDATE TABLE`
- `DELETE TABLE`
- `SELECT SEQUENCE`

If the tables are not created manually before you run the installation with Ant Tasks or the Server Configuration Tool, ensure that you have a user that is able to create the tables. It also needs the following privileges:

- `CREATE INDEX`
- `CREATE SEQUENCE`
- `CREATE TABLE`

For an upgrade of the product, it needs these additional privileges:

- `ALTER TABLE`
- `CREATE VIEW`
- `DROP INDEX`
- `DROP SEQUENCE`
- `DROP TABLE`
- `DROP VIEW`

Database requirements:

The database stores all the data of the MobileFirst Server applications. Before you install the MobileFirst Server components, ensure that the database requirements are met.

DB2 database and user requirements:

Review the database requirement for DB2. Follow the steps to create user, database, and setup your database to meet the specific requirement.

About this task

The page size of the database must be at least 32768.The following procedure creates a database with a page size 32768. It also creates a user (`mfpuser`) and then grants the database access to this user. This user can then be used by the Server Configuration Tool or the Ant tasks to create the tables.

Procedure

1. Create a system user named, for example, `mfpuser` in a DB2 admin group such as `DB2USERS`, by using the appropriate commands for your operating system. Give it a password, for example, `mfpuser`.
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. To create the MobileFirst Server database, enter the SQL statements similar to the following example.

   Replace the user name `mfpuser` with your own.
   ```
   CREATE DATABASE MFPDATA COLLATE USING SYSTEM PAGESIZE 32768
   CONNECT TO MFPDATA
   GRANT CONNECT ON DATABASE TO USER mfpuser
   DISCONNECT MFPDATA
   QUIT
   ```

Oracle database and user requirements:

Review the database requirement for Oracle. Follow the steps to create user, database, and setup your database to meet the specific requirement.

About this task

Ensure that you set the database character set as `Unicode character set (AL32UTF8)` and the national character set as `UTF8 - Unicode 3.0 UTF-8`.

The runtime user (as discussed is "Database users and privileges" on page 6-64) must have an associated table space and enough quota to write the technical data required by the MobileFirst services. For more information about the tables that are used by the product, see "Internal runtime databases" on page 6-315.

The tables are expected to be created in the default schema of the runtime user. The Ant tasks and the Server Configuration Tool create the tables in the default schema of the user passed as argument. For more information about the creation of tables, see "Creating the Oracle database tables manually" on page 6-68.

The procedure creates a database if needed. A user that can create tables and index in this database is added and used as a runtime user.

Procedure

1. If you do not already have a database, use the Oracle Database Configuration Assistant (DBCA) and follow the steps in the wizard to create a new general-purpose database, named `ORCL` in this example:
   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 `MFPUSER`.
     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 `MFPUSER` for the database:

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

MySQL database and user requirements:

Review the database requirement for MySQL. Follow the steps to create user, database, and configure your database to meet the specific requirement.

About this task

Make sure that you set the character set to UTF8.

The following properties must be assigned with appropriate values:

- **`max_allowed_packet`** with 256 M or more

- **innodb_log_file_size** with 250 M or more

For more information about how to set the properties, see MySQL documentation.

The procedure creates a database (MFPDATA) and a user (mfpuser) that can connect to the database with all privileges from a host (mfp-host).

Procedure

1. Run a MySQL command line client with the option `-u root`.
2. Enter the following commands:
   ```
   CREATE DATABASE MFPDATA CHARACTER SET utf8 COLLATE utf8_general_ci;
   GRANT ALL PRIVILEGES ON MFPDATA.* TO 'mfpuser'@'mfp-host' IDENTIFIED BY 'mfpuser-password';
   GRANT ALL PRIVILEGES ON MFPDATA.* TO 'mfpuser'@'localhost' IDENTIFIED BY 'mfpuser-password';
   FLUSH PRIVILEGES;
   ```
   Where *mfpuser* before the "at" sign (@) is the user name, *mfpuser-password* after IDENTIFIED BY is its password, and *mfp-host* is the name of the host on which IBM MobileFirst Platform Foundation runs.

   The user must be able to connect to the MySQL server from the hosts that run the Java application server with the MobileFirst Server applications installed.

Create the database tables manually:

The database tables for the MobileFirst Server applications can be created manually, with Ant Tasks, or with the Server Configuration Tool. The topics provide the explanation and details on how to create them manually.

Depending on your choice of the supported database management system (DBMS), select one of the following topics to create the database tables manually.

Creating the DB2 database tables manually:

Use the SQL scripts that are provided in the MobileFirst Server installation to create the DB2 database tables.

Before you begin

The DB2 database must fulfill the requirement as described in “DB2 database and user requirements” on page 6-65.

About this task

As described in “Setting up databases” on page 6-63, all the four MobileFirst Server components need tables. They can be created in the same schema or in different schemas. However, some constraints apply depending on how the MobileFirst Server applications are deployed to the Java application server. They are the similar to the topic about the possible users for DB2 as described in “Database users and privileges” on page 6-64.

Installation with the Server Configuration Tool
: The same schema is used for all components (MobileFirst Server administration service, MobileFirst Server live update service, MobileFirst Server push service, and MobileFirst runtime)

Installation with Ant tasks
: The sample Ant files that are provided in the product distribution use the same schema for all components. However, it is possible to modify the Ant files to have different schemas:

- The same schema for the administration service and the live update service as they cannot be installed separately with Ant tasks.
- A different schema for the runtime
- A different schema for the push service.

Manual installation
It is possible to assign a different data source, and thus a different schema, to each of the MobileFirst Server components.

The scripts to create the tables are as follows:

- For the administration service, in *mfp_install_dir*/MobileFirstServer/databases/create-mfp-admin-db2.sql.
- For the live update service, in *mfp_install_dir*/MobileFirstServer/databases/create-configservice-db2.sql.
- For the runtime component, in *mfp_install_dir*/MobileFirstServer/databases/create-runtime-db2.sql.
- For the push service, in *mfp_install_dir*/PushService/databases/create-push-db2.sql.

The following procedure creates the tables for all the applications in the same schema (MFPSCM). It assumes that a database and a user are already created. For more information, see "DB2 database and user requirements" on page 6-65.

Procedure

Run DB2 with the following commands with the user (mfpuser):

```
db2 CONNECT TO MFPDATA
db2 SET CURRENT SCHEMA = 'MFPSCM'
db2 -vf mfp_install_dir/MobileFirstServer/databases/create-mfp-admin-db2.sql
db2 -vf mfp_install_dir/MobileFirstServer/databases/create-configservice-db2.sql -t
db2 -vf mfp_install_dir/MobileFirstServer/databases/create-runtime-db2.sql -t
db2 -vf mfp_install_dir/PushService/databases/create-push-db2.sql -t
```

If the tables are created by mfpuser, this user has the privileges on the tables automatically and can use them at run time. If you want to restrict the privileges of the runtime user as described in "Database users and privileges" on page 6-64 or a finer control of privileges, refer to the DB2 documentation.

Creating the Oracle database tables manually:

Use the SQL scripts that are provided in the MobileFirst Server installation to create the Oracle database tables.

Before you begin

The Oracle database must fulfill the requirement as described in "Oracle database and user requirements" on page 6-65.

About this task

As described in "Setting up databases" on page 6-63, all the four MobileFirst Server components need tables. They can be created in the same schema or in different schemas. However, some constraints apply depending on how the MobileFirst Server applications are deployed to the Java application server. The details are described in "Database users and privileges" on page 6-64.

The tables must be created in the default schema of the runtime user. The scripts to create the tables are as follows:

- For the administration service, in *mfp_install_dir*/MobileFirstServer/databases/create-mfp-admin-oracle.sql.
- For the live update service, in *mfp_install_dir*/MobileFirstServer/databases/create-configservice-oracle.sql.
- For the runtime component, in *mfp_install_dir*/MobileFirstServer/databases/create-runtime-oracle.sql.
- For the push service, in *mfp_install_dir*/PushService/databases/create-push-oracle.sql.

The following procedure creates the tables for all the applications for the same user (MFPUSER). It assumes that a database and a user are already created. For more information, see "Oracle database and user requirements" on page 6-65.

Procedure

Run the following commands in Oracle SQLPlus:

```
CONNECT MFPUSER/MFPUSER_password@ORCL
@mfp_install_dir/MobileFirstServer/databases/create-mfp-admin-oracle.sql
@mfp_install_dir/MobileFirstServer/databases/create-configservice-oracle.sql
@mfp_install_dir/MobileFirstServer/databases/create-runtime-oracle.sql
@mfp_install_dir/PushService/databases/create-push-oracle.sql
DISCONNECT;
```

If the tables are created by MFPUSER, this user has the privileges on the tables automatically and can use them at run time. The tables are created in the user's default schema. If you want to restrict the privileges of the runtime user as described in "Database users and privileges" on page 6-64 or have a finer control of privileges, refer to the Oracle documentation.

Creating the MySQL database tables manually:

Use the SQL scripts that are provided in the MobileFirst Server installation to create the MySQL database tables.

Before you begin

The MySQL database must fulfill the requirement as described in "MySQL database and user requirements" on page 6-66.

About this task

As described in "Setting up databases" on page 6-63, all the four MobileFirst Server components need tables. They can be created in the same schema or in different schemas. However, some constraints apply depending on how the MobileFirst Server applications are deployed to the Java application server. They are the similar to the topic about the possible users for MySQL as described in "Database users and privileges" on page 6-64.

Installation with the Server Configuration Tool
The same database is used for all components (MobileFirst Server administration service, MobileFirst Server live update service, MobileFirst Server push service, and MobileFirst runtime)

Installation with Ant tasks

The sample Ant files that are provided in the product distribution use the same database for all components. However, it is possible to modify the Ant files to have different database:

- The same database for the administration service and the live update service as they cannot be installed separately with Ant tasks.
- A different database for the runtime
- A different database for the push service.

Manual installation

It is possible to assign a different data source, and thus a different database, to each of the MobileFirst Server components.

The scripts to create the tables are as follows:

- For the administration service, in *mfp_install_dir*/MobileFirstServer/databases/create-mfp-admin-mysql.sql.
- For the live update service, in *mfp_install_dir*/MobileFirstServer/databases/create-configservice-mysql.sql.
- For the runtime component, in *mfp_install_dir*/MobileFirstServer/databases/create-runtime-mysql.sql.
- For the push service, in *mfp_install_dir*/PushService/databases/create-push-mysql.sql.

The following example creates the tables for all the applications for the same user and database. It assumes that a database and a user has been created as in 'Requirements for the databases/MySQL'

The following procedure creates the tables for all the applications for the same user (mfpuser) and database (MFPDATA). It assumes that a database and a user are already created. For more information, see “MySQL database and user requirements” on page 6-66.

Procedure

1. Run a MySQL command line client with the option: -u mfpuser.
2. Enter the following commands:

```
USE MFPDATA;
SOURCE mfp_install_dir/MobileFirstServer/databases/create-mfp-admin-mysql.sql;
SOURCE mfp_install_dir/MobileFirstServer/databases/create-configservice-mysql.sql;
SOURCE mfp_install_dir/MobileFirstServer/databases/create-runtime-mysql.sql;
SOURCE mfp_install_dir/PushService/databases/create-push-mysql.sql;
```

Create the database tables with the Server Configuration Tool:

The database tables for the MobileFirst Server applications can be created manually, with Ant Tasks, or with the Server Configuration Tool. The topics provide the explanation and details about database setup when you install MobileFirst Server with the Server Configuration Tool.

The Server Configuration Tool can create the database tables as part of the installation process. In some cases, it can even create a database and a user for the MobileFirst Server components. For an overview of the installation process with theServer Configuration Tool, see “Installing MobileFirst Server in graphical mode” on page 6-5.

After you complete the configuration credentials and click **Deploy** in the Server Configuration Tool pane, the following operations are run:

- Create the database and user if needed.
- Verify whether the MobileFirst Server tables exist in the database. If they do not exist, create the tables.
- Deploys the MobileFirst Server applications to the application server.

If the database tables are created manually before you run the Server Configuration Tool, the tool can detect them and skip the phase of setting up the tables.

Depending on your choice of the supported database management system (DBMS), select one of the following topics for more details on how the tool creates the database tables.

Creating the DB2 database tables with the Server Configuration Tool:

Use the Server Configuration Tool that is provided with MobileFirst Server installation to create the DB2 database tables.

About this task

The Server Configuration Tool can create a database in the default DB2 instance. In **Database Selection** panel of the Server Configuration Tool, select the `IBM DB2` option. In the next three panes, enter the database credentials. If the database name that is entered in the **Database Additional Settings** panel does not exist in the DB2 instance, you can enter additional information to enable the tool to create a database for you.

The following procedure provides some extra steps that you need to do when you create the database table with the tool.

Procedure

1. Run an SSH server on the computer that runs the DB2 database.

   The Server Configuration Tool opens an SSH session to the DB2 host to create the database. Except on Linux and some versions of UNIX systems, the SSH server is needed even if the DB2 database runs on the same computer as the Server Configuration Tool.
2. In the **Database creation request** panel, enter the login ID and password of a DB2 user with administration privileges (`SYSADM` or `SYSCTRL` permissions).

   You need to provide this user with the administration privileges if the DB2 user that is entered in the **Database Additional Settings** panel does not have those permissions.

Results

The Server Configuration Tool creates the database tables with default settings with the following SQL statement:

```
CREATE DATABASE MFPDATA COLLATE USING SYSTEM PAGESIZE 32768
```

It is not meant to be used for production as in a default DB2 installation, many privileges are granted to PUBLIC.

Creating the Oracle database tables with the Server Configuration Tool:

Use the Server Configuration Tool that is provided with MobileFirst Server installation to create the Oracle database tables.

About this task

In **Database Selection** panel of the Server Configuration Tool, select the `Oracle Standard or Enterprise Editions, 11g or 12c` option. In the next three panes, enter the database credentials.

When you enter the Oracle user name in **Database Additional Settings** panel, it must be in uppercase. If you have an Oracle database user (FOO), but you enter a user name with lowercase (`foo`), the Server Configuration Tool considers it as another user. Unlike other tools for Oracle database, the Server Configuration Tool protects the user name against automatic conversion to uppercase.

The Server Configuration Tool uses a service name or Oracle System Identifier (SID) to identify a database. However, if you want to make the connection to Oracle RAC, you need to enter a complex JDBC URL. In this case, in the **Database Settings** panel, select the **Connect using generic Oracle JDBC URLs** option and enter a URL for the Oracle thin driver.

If you need to create database and user for Oracle, use the Oracle Database Creation Assistant (DBCA) tool. For more information, see "Oracle database and user requirements" on page 6-65.

The Server Configuration Tool can do the same but with a limitation. The tool can create a user for Oracle 11g or Oracle 12g. However, it can create a database only for Oracle 11g, and not for Oracle 12c.

If the database name or user name that is entered in the **Database Additional Settings** panel does not exist, refer to the following two sections for the extra steps to create the database or the user.

Creating the database:
Procedure

1. Run an SSH server on the computer that runs the Oracle database.

   The Server Configuration Tool opens an SSH session to the Oracle host to create the database. Except on Linux and some versions of UNIX systems, the SSH server is needed even if the Oracle database runs on the same computer as the Server Configuration Tool.
2. In **Database creation request** panel, enter the login ID and password of an Oracle database user that has the privileges to create a database.
3. In the same panel, also enter the password for the `SYS` user and the `SYSTEM` user for the database that is to be created.

Results

A database is created with the SID name that is entered in the **Database Additional Settings** panel. It is not meant to be used for production.

Creating the user:
Procedure

1. Run an SSH server on the computer that runs the Oracle database.

The Server Configuration Tool opens an SSH session to the Oracle host to create the database. Except on Linux and some versions of UNIX systems, the SSH server is needed even if the Oracle database runs on the same computer as the Server Configuration Tool.

2. In the **Database Additional Settings** panel, enter the login ID and password of the database user that is to be created.
3. In **Database creation request** panel, enter the login ID and password of an Oracle database user that has the privileges to create a database user.
4. In same panel, also enter the password for the `SYSTEM` user of the database.

Results

A database user is created with the name and password that are entered in the **Database Additional Settings** panel.

Creating the MySQL database tables with the Server Configuration Tool:

Use the Server Configuration Tool that is provided with MobileFirst Server installation to create the MySQL database tables.

About this task

The Server Configuration Tool can create a MySQL database for you. In **Database Selection** panel of the Server Configuration Tool, select the `MySQL 5.5.x, 5.6.x or 5.7.x` option. In the next three panes, enter the database credentials. If the database or the user that you enter in the **Database Additional Settings** panel does not exist, the tool can create it.

If MySQL server does not have the settings that are recommended in "MySQL database and user requirements" on page 6-66, the Server Configuration Tool displays a warning. Make sure to fulfill the requirements before you run the Server Configuration Tool.

The following procedure provides some extra steps that you need to do when you create the database tables with the tool.

Procedure

1. In the **Database Additional Settings** panel, besides the connection settings, you must enter all the hosts from which the user is allowed to connect to the database. That is, all the hosts where MobileFirst Server runs.
2. In the **Database creation request** panel, enter the login ID and the password of a MySQL administrator. By default, the administrator is root.

Create the database tables with Ant tasks:

The database tables for the MobileFirst Server applications can be created manually, with Ant Tasks, or with the Server Configuration Tool. The topics provide the explanation and details on how to create them with Ant tasks.

You can find relevant information in this section about the setting up of the database if MobileFirst Server is installed with Ant Tasks.

You can use Ant Tasks to set up the MobileFirst Server database tables. In some cases, you can also create a database and a user with these tasks. For an overview of the installation process with Ant Tasks, see "Installing MobileFirst Server in command line mode" on page 6-22.

A set of sample Ant files is provided with the installation to help you get started with the Ant tasks. You can find the files in `mfp_install_dir/MobileFirstServer/configurations-samples`. The files are named after the following patterns:

`configure-<appserver>-<dbms>.xml`
The Ant files can do these tasks:
- Create the tables in a database if the database and database user exist. The requirements for the database are listed in "Database requirements" on page 6-65.
- Deploy the WAR files of the MobileFirst Server components to the application server. These Ant files use the same database user to create the tables, and to install the run time database user for the applications at run time. The files also use the same database user for all theMobileFirst Server applications.

`create-database-<dbms>.xml`
The Ant files can create a database if needed on the supported database management system (DBMS), and then create the tables in the database. However, as the database is created with default settings, it is not meant to be used for production.

In the Ant files, you can find the predefined targets that use **`configureDatabase`** Ant task to set up the database. For more information, see "Ant **`configuredatabase`** task reference" on page 6-268.

Using the sample Ant files

The sample Ant files have predefined targets. Follow this procedure to use the files.
1. Copy the Ant file according to your application server and database configuration in a working directory.
2. Edit the file and enter the values for your configuration in the `<! -- Start of Property Parameters -->` section for the Ant file.
3. Run the Ant file with the `databases` target: `mfp_install_dir/shortcuts/ant -f your_ant_file databases`.

This command creates the tables in the specified database and schema for all MobileFirst Server applications (MobileFirst Server administration service, MobileFirst Server live update service, MobileFirst Server push service, and MobileFirst Server runtime). A log for the operations is produced and stored in your disk.
- On Windows, it is in `C:\Users\user_name\Documents\IBM MobileFirst Platform Server Data\Configuration Logs\` directory.
- On UNIX, it is in `$HOME/.mobilefirst_platform_server/configuration-logs/` directory.

Different users for the database tables creation and for run time

The sample Ant files in `mfp_install_dir/MobileFirstServer/configurations-samples` use the same database user for:

- All the MobileFirst Server applications (the administration service, the live update service, the push service, and the runtime)
- The user that is used to create the database and the user at run time for the data source in the application server.

If you want to separate the users as described in "Database users and privileges" on page 6-64, you need to create your own Ant file, or modify the sample Ant files so that each database target has a different user. For more information, see "Installation reference" on page 6-268.

For DB2 and MySQL, it is possible to have different users for the database creation and for the run time. The privileges for each type of the users are listed in "Database users and privileges" on page 6-64. For Oracle, you cannot have a different user for database creation and for the run time. The Ant tasks consider that the tables are in the default schema of a user. If you want to reduce privileges for the runtime user, you must create the tables manually in the default schema of the user that will be used at run time. For more information, see "Creating the Oracle database tables manually" on page 6-68.

Depending on your choice of the supported database management system (DBMS), select one of the following topics to create the database with Ant tasks.

Creating the DB2 database tables with Ant tasks:

Use Ant tasks that are provided with MobileFirst Server installation to create the DB2 database.

To create the database tables in a database that already exists, see "Create the database tables with Ant tasks" on page 6-73.

To create a database and the database tables, you can do so by Ant tasks. The Ant tasks create a database in the default instance of DB2 if you use an Ant file that contains the `dba` element. This element can be found in the sample Ant files named as `create-database-<`*`dbms`*`>.xml`.

Before you run the Ant tasks, make sure that you have an SSH server on the computer that runs the DB2 database. The **`configureDatabase`** Ant task opens an SSH session to the DB2 host to create the database. The SSH server is needed even if the DB2 database runs on the same computer where you run the Ant tasks (except on Linux and some versions of UNIX systems).

Follow the general guidelines as described in "Create the database tables with Ant tasks" on page 6-73 to edit the copy of the `create-database-db2.xml` file.

You must also provide the login ID and password of a DB2 user with administration privileges (`SYSADM` or `SYSCTRL` permissions) in the `dba` element. In the sample Ant file for DB2 (`create-database-db2.xml`), the properties to set are: **`database.db2.admin.username`** and **`database.db2.admin.password`**.

When the `databases` Ant target is called, the **`configureDatabase`** Ant task creates a database with default settings with the following SQL statement:

```
CREATE DATABASE MFPDATA COLLATE USING SYSTEM PAGESIZE 32768
```

It is not meant to be used for production as in a default DB2 installation, many privileges are granted to PUBLIC.

Creating the Oracle database tables with Ant tasks:

Use Ant tasks that are provided with MobileFirst Server installation to create the Oracle database tables.

About this task

When you enter the Oracle user name in Ant file, it must be in uppercase. If you have an Oracle database user (FOO), but you enter a user name with lowercase (`foo`), the **`configureDatabase`** Ant task considers it as another user. Unlike other tools for Oracle database, the **`configureDatabase`** Ant task protects the user name against automatic conversion to uppercase.

The **`configureDatabase`** Ant task uses a service name or Oracle System Identifier (SID) to identify a database. However, if you want to make the connection to Oracle RAC, you need to enter a complex JDBC URL. In this case, the `oracle` element that is within the **`configureDatabase`** Ant task must use the attributes (**`url`**, **`user`**, and **`password`**) instead of these attributes (**`database`**, **`server`**, **`port`**, **`user`**, and **`password`**) attributes. For more information, see Table 6-69 on page 6-274 in "Ant **`configuredatabase`** task reference" on page 6-268. The sample Ant files in *`mfp_install_dir`*`/MobileFirstServer/configurations-samples` use the **`database`**, **`server`**, **`port`**, **`user`**, and **`password`** attributes in the `oracle` element. They must be modified if you need to connect to Oracle with a JDBC URL.

To create the database tables in a database that already exists, see "Create the database tables with Ant tasks" on page 6-73.

To create a database, user, or the database tables, use the Oracle Database Creation Assistant (DBCA) tool. For more information, see "Oracle database and user requirements" on page 6-65.

The **`configureDatabase`** Ant task can do the same but with a limitation. The task can create a database user for Oracle 11g or Oracle 12g. However, it can create a database only for Oracle 11g, and not for Oracle 12c. Refer to the following two sections for the extra steps that you need to create the database or the user.

Creating the database:
Before you begin

Follow the general guidelines as described in "Create the database tables with Ant tasks" on page 6-73 to edit the copy of the `create-database-oracle.xml` file.

Procedure

1. Run an SSH server on the computer that runs the Oracle database.

   The **`configureDatabase`** Ant task opens an SSH session to the Oracle host to create the database. Except on Linux and some versions of UNIX systems, the SSH server is needed even if the Oracle database runs on the same computer where you run the Ant tasks.
2. In `dba` element that is defined in the `create-database-oracle.xml` file, enter the login ID and password of an Oracle database user that can connect to the Oracle Server via SSH and has the privileges to create a database. You can assign the values in the following properties:
   - **`database.oracle.admin.username`**
   - **`database.oracle.admin.password`**

3. In `oracle` element, enter the database name that you want to create. The attribute is **`database`**. You can assign the value in the **`database.oracle.mfp.dbname`** property.
4. In the same `oracle` element, also enter the password for the `SYS` user and the `SYSTEM` user for the database that is to be created. The attributes are **`sysPassword`** and **`systemPassword`**. You can assign the values in the corresponding properties:
   - **`database.oracle.sysPassword`**
   - **`database.oracle.systemPassword`**
5. After all the database credentials are entered in the Ant file, save it and run the `databases` Ant target.

Results

A database is created with the SID name that is entered in the **`database`** of the `oracle` element. It is not meant to be used for production.

Creating the user:
Before you begin

Follow the general guidelines as described in "Create the database tables with Ant tasks" on page 6-73 to edit the copy of the `create-database-oracle.xml` file.

Procedure
1. Run an SSH server on the computer that runs the Oracle database.

   The **`configureDatabase`** Ant task opens an SSH session to the Oracle host to create the database. Except on Linux and some versions of UNIX systems, the SSH server is needed even if the Oracle database runs on the same computer where you run the Ant tasks.
2. In `oracle` element that is defined in the `create-database-oracle.xml` file, enter the login ID and password of an Oracle database user that you want to create. The attributes are **`user`** and **`password`**. You can assign the values in the corresponding properties:
   - **`database.oracle.mfp.username`**
   - **`database.oracle.mfp.password`**
3. In the same `oracle` element, also enter the password for the `SYSTEM` user for the database. The attribute is **`systemPassword`**. You can assign the value in the **`database.oracle.systemPassword`** property.
4. In `dba` element, enter the login ID and password of an Oracle database user that has the privileges to create a user. You can assign the values in the following properties:
   - **`database.oracle.admin.username`**
   - **`database.oracle.admin.password`**
5. After all the database credentials are entered in the Ant file, save it and run the `databases` Ant target.

Results

A database user is created with the name and password that are entered in the `oracle` element. This user has the privileges to create the MobileFirst Server tables, upgrade them and use them at run time.

Creating the MySQL database tables with Ant tasks:

Use Ant Tasks that are provided with MobileFirst Server installation to create the MySQL database tables.

About this task

To create the database tables in a database that already exists, see "Create the database tables with Ant tasks" on page 6-73.

If MySQL server does not have the settings that are recommended in "MySQL database and user requirements" on page 6-66, the **`configureDatabase`** Ant task displays a warning. Make sure to fulfill the requirements before you run the Ant task.

To create a database and the database tables, follow the general guidelines as described in "Create the database tables with Ant tasks" on page 6-73 to edit the copy of the `create-database-mysql.xml` file.

The following procedure provides some extra steps that you need to do when you create the database tables with the **`configureDatabase`** Ant task.

Procedure

1. In `dba` element that is defined in the `create-database-mysql.xml` file, enter the login ID and password of a MySQL administrator. By default, the administrator is root. You can assign the values in the following properties:
   - **`database.mysql.admin.username`**
   - **`database.mysql.admin.password`**
2. In the `mysql` element, add a `client` element for each host from which the user is allowed to connect to the database. That is, all the hosts where MobileFirst Server runs.
3. After all the database credentials are entered in the Ant file, save it and run the `databases` Ant target.

Topologies and network flows

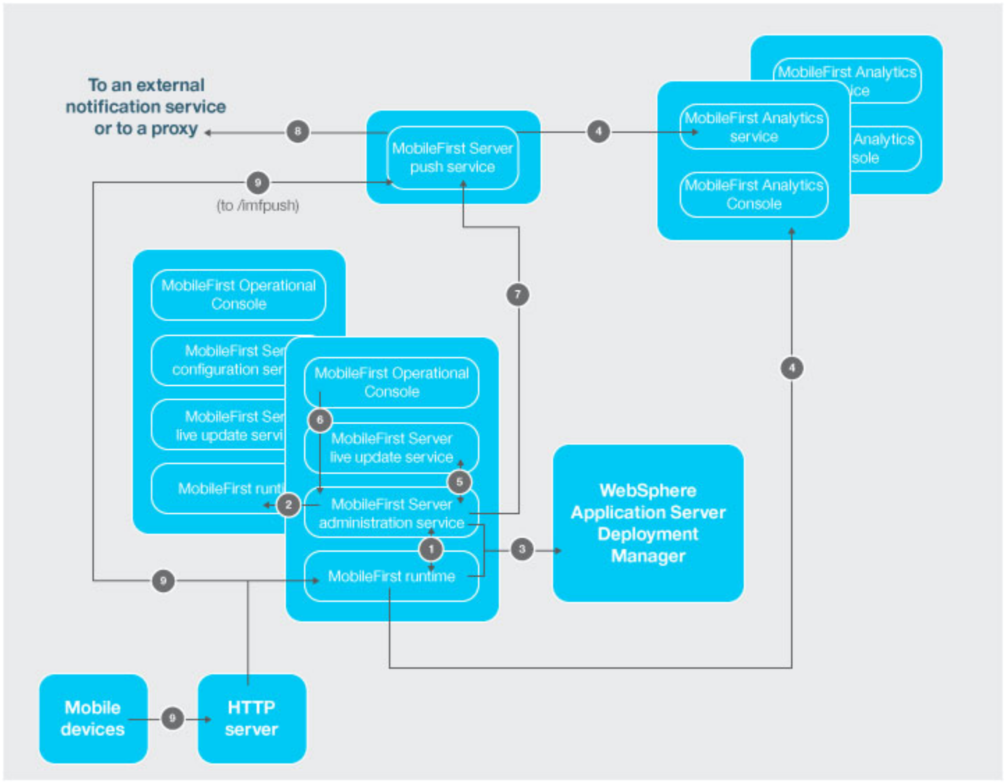
Topics about possible server topologies for MobileFirst Server components and the network flows.

The components are deployed according to the server topology that you use. The network flows topic also explains to you how the components communicate with one another and with the devices.

Network flows between the MobileFirst Server components:

The MobileFirst Server components can communicate with each other over JMX or HTTP. You need to configure certain JNDI properties to enable the communications.

The network flows between the components and the device can be illustrated by the following image:


To an external notification service or to a proxy
8
9
(to /imfpush)
MobileFirst Server push service
4
MobileFirst Analytics service
MobileFirst Analytics Console
7
MobileFirst Operational Console
MobileFirst Server configuration service
MobileFirst Server live update service
MobileFirst runtime
6
MobileFirst Server live update service
5
2
MobileFirst Server administration service
1
3
MobileFirst runtime
WebSphere Application Server Deployment Manager
Mobile devices
HTTP server

The flows between the various MobileFirst Server components, IBM MobileFirst Analytics, the mobile devices, and the application server are explained in the following sections:

1. "MobileFirst runtime to MobileFirst Server administration service"
2. "MobileFirst Server administration service to MobileFirst runtime in other servers" on page 6-81
3. "MobileFirst Server administration service and MobileFirst runtime to the deployment manager on WebSphere Application Server Network Deployment" on page 6-81
4. "MobileFirst Server push service and MobileFirst runtime to MobileFirst Analytics" on page 6-82
5. "MobileFirst Server administration service to MobileFirst Server live update service" on page 6-82
6. "MobileFirst Operations Console to MobileFirst Server administration service" on page 6-83
7. "MobileFirst Server administration service to MobileFirst Server push service, and to the authorization server " on page 6-83
8. " MobileFirst Server push service to an external push notification service (outbound) " on page 6-84
9. " Mobile devices to MobileFirst runtime " on page 6-84

MobileFirst runtime to MobileFirst Server administration service

The runtime and the administration service can communicate with each other through JMX and HTTP. This communication occurs during the initialization phase of the runtime. The runtime contacts the administration service local to its application server to get the list of the adapters and applications that it needs to serve. The communication also happens when some administration operations are run from MobileFirst Operations Console or the administration service. On WebSphere Application Server Network Deployment, the runtime can contact an administration service that is installed on another server of the cell. This enables the non-symmetric deployment (see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84). However, on all other application servers (Apache Tomcat, WebSphere Application Server Liberty, or stand-alone WebSphere Application Server), the administration service must be running on the same server as the runtime.

The protocols for JMX depend on the application server:

- Apache Tomcat - RMI
- WebSphere Application Server Liberty - HTTPS (with the REST connector)
- WebSphere Application Server - SOAP or RMI

For the communication via JMX, it is required that these protocols are available on the application server. For more information about the requirements, see "Application server prerequisites" on page 6-100.

The JMX beans of the runtime and the administration service are obtained from the application server. However, in the case of WebSphere Application Server Network Deployment, the JMX beans are obtained from the deployment manager. The deployment manager has the view of all the beans of a cell on WebSphere Application Server Network Deployment. As such, some configurations are not needed on WebSphere Application Server Network Deployment (such as the farm configuration), and non-symmetric deployment is possible on WebSphere Application Server Network Deployment. For more information, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84.

To distinguish different installation of MobileFirst Server on the same application server or on the same WebSphere Application Server cell, you can use an environment ID, which is a JNDI variable. By default, this variable has an empty value. A runtime with a given environment ID communicates only with an administration service that has the same environment ID. For example, the administration service has an environment ID set to `X`, and the runtime has a different environment ID (for example, `Y`), then the two components do not see each other. The MobileFirst Operations Console shows no runtime available.

An administration service must be able to communicate with all the MobileFirst runtime components of a cluster. When an administration operation is run, such as uploading a new version of an adapter, or changing the active status of an application, all runtime components of the cluster must be notified about the change. If the application server is not WebSphere Application Server Network Deployment, this communication can happen only if a farm is configured. For more information, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84.

The runtime also communicates with the administration service through HTTP or HTTPS to download large artifacts such as the adapters. A URL is generated by the administration service and the runtime opens and outbound HTTP or HTTPS connection to request an artifact from this URL. It is possible to override the default URL generation by defining the JNDI properties (**`mfp.admin.proxy.port`**, **`mfp.admin.proxy.protocol`**, and **`mfp.admin.proxy.host`**) in the administration service. The administration service might also need to communicate with the runtime through HTTP or HTTPS to get the OAuth tokens that are used to run the push operations. For more information, see "MobileFirst Server administration service to MobileFirst Server push service, and to the authorization server " on page 6-83.

The JNDI properties that are used for the communication between the runtime and the administration service are as follows:

MobileFirst Server administration service

- Table 6-30 on page 6-174 - JNDI properties for administration services: JMX
- Table 6-33 on page 6-177 - JNDI properties for administration services: proxies
- Table 6-34 on page 6-177 - JNDI properties for administration services: topologies

MobileFirst runtime

"List of JNDI properties for MobileFirst runtime" on page 6-183

MobileFirst Server administration service to MobileFirst runtime in other servers

As described in "MobileFirst runtime to MobileFirst Server administration service" on page 6-79, it is required to have the communication between an administration service and all the runtime components of a cluster. When an administration operation is run, all the runtime components of a cluster can then be notified about this modification. The communication is through JMX.

On WebSphere Application Server Network Deployment, this communication can occur without any specific configuration. All the JMX MBeans that correspond to the same environment ID are obtained from the deployment manager.

For a cluster of stand-alone WebSphere Application Server, WebSphere Application Server Liberty profile, or Apache Tomcat, the communication can happen only if a farm is configured. For more information, see "Installing a server farm" on page 6-139.

MobileFirst Server administration service and MobileFirst runtime to the deployment manager on WebSphere Application Server Network Deployment

On WebSphere Application Server Network Deployment, the runtime and the administration service obtain the JMX MBeans that are used in "MobileFirst runtime to MobileFirst Server administration service" on page 6-79 and "MobileFirst Server administration service to MobileFirst runtime in other servers" by communicating with the deployment manager. The corresponding JNDI properties are **`mfp.admin.jmx.dmgr.*`** in JNDI properties for administration services: JMX.

The deployment manager must be running to allow the operations that require JMX communication between the runtime and the administration service. Such operations can be a runtime initialization, or the notification of a modification performed through the administration service.

MobileFirst Server push service and MobileFirst runtime to MobileFirst Analytics

The runtime sends data to MobileFirst Analytics through HTTP or HTTPS. The JNDI properties of the runtime that are used to define this communication are:

- **`mfp.analytics.url`**- the URL that is exposed by MobileFirst Analytics service to receive incoming analytics data from the runtime. Example:
  `http://<hostname>:<port>/analytics-service/rest`

  When MobileFirst Analytics is installed as a cluster, the data can be sent to any member of the cluster.
- **`mfp.analytics.username`** - the user name that is used to access MobileFirst Analytics service. The analytics service is protected by a security role.
- **`mfp.analytics.password`** - the password to access the analytics service.
- **`mfp.analytics.console.url`** - the URL that is passed to MobileFirst Operations Console to display a link to MobileFirst Analytics Console. Example:
  `http://<hostname>:<port>/analytics/console`

The JNDI properties of the push service that are used to define this communication are:

- **`mfp.push.analytics.endpoint`** - the URL that is exposed by MobileFirst Analytics service to receive incoming analytics data from the push service. Example:
  `http://<hostname>:<port>/analytics-service/rest`

  When MobileFirst Analytics is installed as a cluster, the data can be sent to any member of the cluster.
- **`mfp.push.analytics.username`** - the user name that is used to access MobileFirst Analytics service. The analytics service is protected a security role.
- **`mfp.push.analytics.password`** - the password to access the analytics service.

MobileFirst Server administration service to MobileFirst Server live update service

The administration service communicates with the live update service to store and retrieve configuration information about the MobileFirst artifacts. The communication is performed through HTTP or HTTPS.

The URL to contact the live update service is automatically generated by the administration service. Both services must be on the same application server. The context root of the live update service must define in this way: `<adminContextRoot>config`. For example, if the context root of the administration service is `mfpadmin`, then the context root of the live update service must be `mfpadminconfig`. It is possible to override the default URL generation by defining the JNDI properties (**`mfp.admin.proxy.port`**, **`mfp.admin.proxy.protocol`**, and **`mfp.admin.proxy.host`**) in the administration service.

The JNDI properties to configure this communication between the two services are:

- **`mfp.config.service.user`**
- **`mfp.config.service.password`**
- And those properties in JNDI properties for administration services: proxies.

MobileFirst Operations Console to MobileFirst Server administration service

MobileFirst Operations Console is a web user interface and acts as the front end to the administration service. It communicates with the REST services of the administration service through HTTP or HTTPS. The users who are allowed to use the console, must also be allowed to use the administration service. Each user that is mapped to a certain security role of the console must also be mapped to the same security role of the service. With this setup, the requests from the console can thus be accepted by the service.

The JNDI properties to configure this communication are in JNDI properties for the MobileFirst Operations Console.

Note: The **`mfp.admin.endpoint`** property enables the console to locate the administration service. You can use the asterisk (*) character as wildcard for specifying that the URL, generated by the console to contact the administration services, use the same value as the incoming HTTP request to the console. For example: `*://*:*/mfpadmin` means use the same protocol, host, and port as the console, but use `mfpadmin` as context root. This property is specified for the console application.

MobileFirst Server administration service to MobileFirst Server push service, and to the authorization server

The administration service communicates with the push service to request various push operations. This communication is secured through the OAuth protocol. Both services need to be registered as confidential clients. An initial registration can be performed at installation time. In this process, both services need to contact an authorization server. This authorization server can be MobileFirst runtime.

The JNDI properties of the administration service to configure this communication are:

- **`mfp.admin.push.url`** - the URL of the push service.
- **`mfp.admin.authorization.server.url`** - the URL of the MobileFirst authorization server.
- **`mfp.admin.authorization.client.id`** - the client ID of the administration service, as an OAuth confidential client.
- **`mfp.admin.authorization.client.secret`** - the secret code that is used to get the OAuth-based tokens.

Note: The **`mfp.push.authorization.client.id`** and **`mfp.push.authorization.client.secret`** properties of the administration service can be used to register the push service automatically as a confidential client when the administration service starts. The push service must be configured with the same values.
The JNDI properties of the push service to configure this communication are:

- **`mfp.push.authorization.server.url`** - the URL of the MobileFirst authorization server. Same as the property **`mfp.admin.authorization.server.url`**.
- **`mfp.push.authorization.client.id`** - the client ID of the push service to contact the authorization server.
- **`mfp.push.authorization.client.secret`** - the secret code that is used to contact the authorization server.

MobileFirst Server push service to an external push notification service (outbound)

The push service generates outbound traffic to the external notification service such as Apple Push Notification Service (APNS) or Google Cloud Messaging (GCM). This communication can also be done through a proxy. Depending on the notification service, the following JNDI properties must be set:
- **`push.apns.proxy.*`**
- **`push.gcm.proxy.*`**

For more information, see "List of JNDI properties for MobileFirst Server push service" on page 6-186.

Mobile devices to MobileFirst runtime

The mobile devices contact the runtime. The security of this communication is determined by the configuration of the application and the adapters that are requested. For more information, see "MobileFirst security framework" on page 7-265.

Constraints on the MobileFirst Server components and MobileFirst Analytics:

Understand the constraints on the various MobileFirst Server components and MobileFirst Analytics before you decide your server topology.

For more in-depth explanation about the server topology for the various MobileFirst Server components, see the following topics.

Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime:

Find out the constraints and the deployment mode of the administration service, live update service, and the runtime per server topology.

The live update service must be always installed with the administration service on the same application server as explained in "MobileFirst Server administration service to MobileFirst Server live update service" on page 6-82. The context root of the live update service must define in this way: `/<adminContextRoot>config`. For example, if the context root of the administration service is `/mfpadmin`, then the context root of the live update service must be `/mfpadminconfig`.

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
- Liberty collective

Modes of deployment

Depending on the application server topology that you use, you have two modes of deployment choice for deploying the administration service, the live update service and the runtime in the application server infrastructure. In asymmetric deployment, you can install the runtimes on different application servers from the administration and the live update services.

Symmetric deployment
In symmetrical deployment, you must install the MobileFirst administration components (MobileFirst Operations Console, the administration service, and the live update service applications) and the runtime on the same application server.

Asymmetric deployment
In asymmetric deployment, you can install the runtimes on different application servers from the MobileFirst administration components.

Asymmetric deployment is only supported for WebSphere Application Server Network Deployment cell topology and for Liberty collective 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).

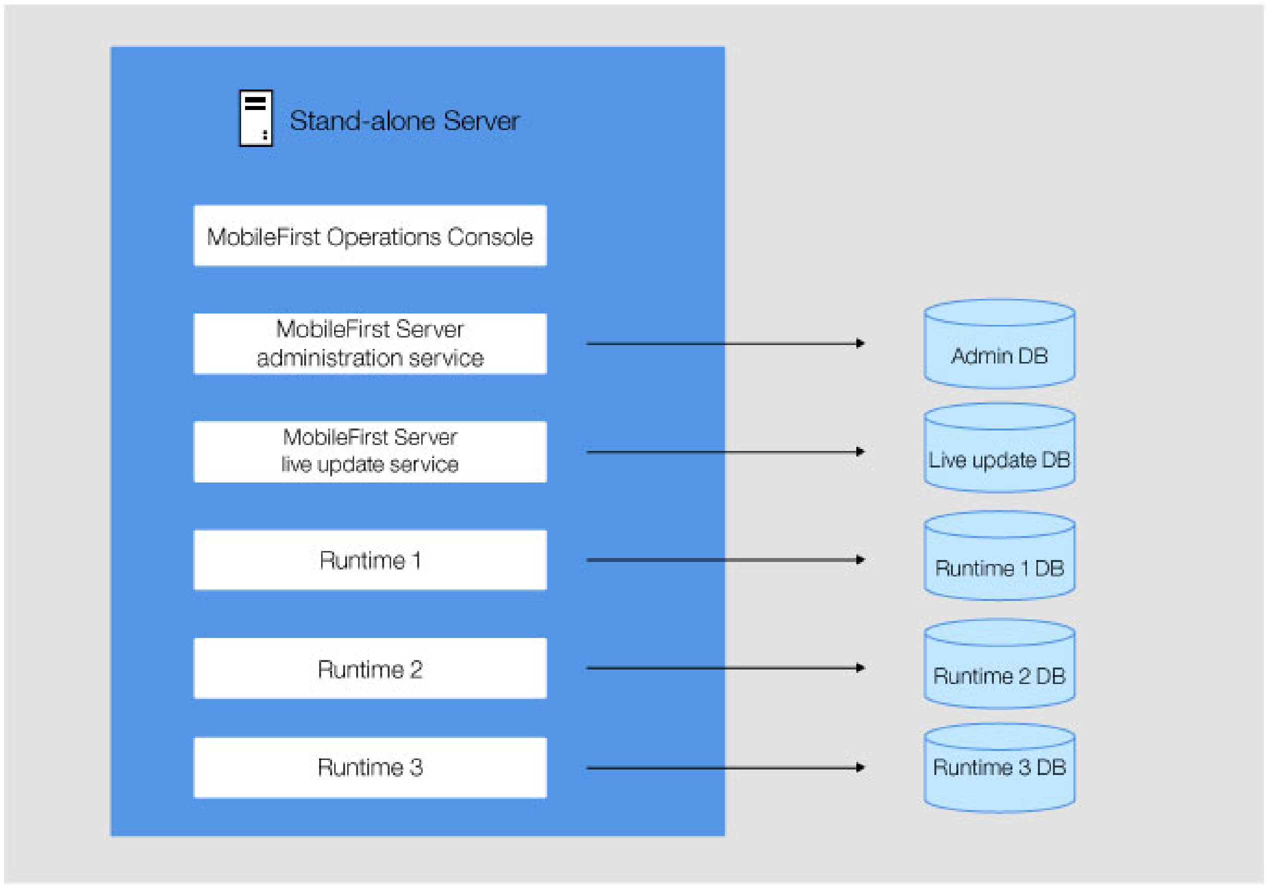

Stand-alone Server
MobileFirst Operations Console
MobileFirst Server administration service
MobileFirst Server live update service
Runtime 1
Runtime 2
Runtime 3
Admin DB
Live update DB
Runtime 1 DB
Runtime 2 DB
Runtime 3 DB

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 and one live update 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 live update service uses its own live update 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 service" on page 6-174 and "List of JNDI properties for MobileFirst runtime" on page 6-183

Stand-alone WebSphere Application Server Liberty profile server

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

Table 6-11. Global JNDI properties for administration services and runtimes in WebSphere Application Server Liberty stand-alone topology.

JNDI properties	Values
`mfp.topology.platform`	`Liberty`
`mfp.topology.clustermode`	`Standalone`
`mfp.admin.jmx.host`	The host name of the WebSphere Application Server Liberty profile server.
`mfp.admin.jmx.port`	The 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.
`mfp.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.
`mfp.admin.jmx.pwd`	The 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 **`mfp.admin.environmentid`** JNDI property.

- On each runtime, the same value for the local **`mfp.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 the runtimes.

Table 6-12. Local JNDI properties for administration services and runtimes in Apache Tomcat stand-alone topology.

JNDI properties	Values
`mfp.topology.platform`	`Tomcat`
`mfp.topology.clustermode`	`Standalone`

JVM properties are also required to define Java Management Extensions (JMX) Remote Method Invocation (RMI). For more information, see "Configuring JMX connection for Apache Tomcat" on page 6-101.

If the Apache Tomcat server is running behind a firewall, the **`mfp.admin.rmi.registryPort`** and **`mfp.admin.rmi.serverPort`** JNDI properties are required for the administration service. See "Configuring JMX connection for Apache Tomcat" on page 6-101.

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 **`mfp.admin.environmentid`** JNDI property.
- On each runtime, the same value for the local **`mfp.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 the runtimes.

Table 6-13. Local JNDI properties for administration services and runtimes in WebSphere Application Server stand-alone topology.

JNDI properties	Values
`mfp.topology.platform`	`WAS`
`mfp.topology.clustermode`	`Standalone`
`mfp.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 **`mfp.admin.environmentid`** JNDI property.

- On each runtime, the same value for the local `mfp.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 service 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. For example, a set of several Liberty servers can be configured as a server farm. Conversely, a mix of Liberty server, Tomcat server, or stand-alone WebSphere Application Server cannot be configured as a server farm.

In this topology, all the administration components (MobileFirst Operations Console, the administration service, and the live update service) and the runtimes are deployed on every server in the farm.

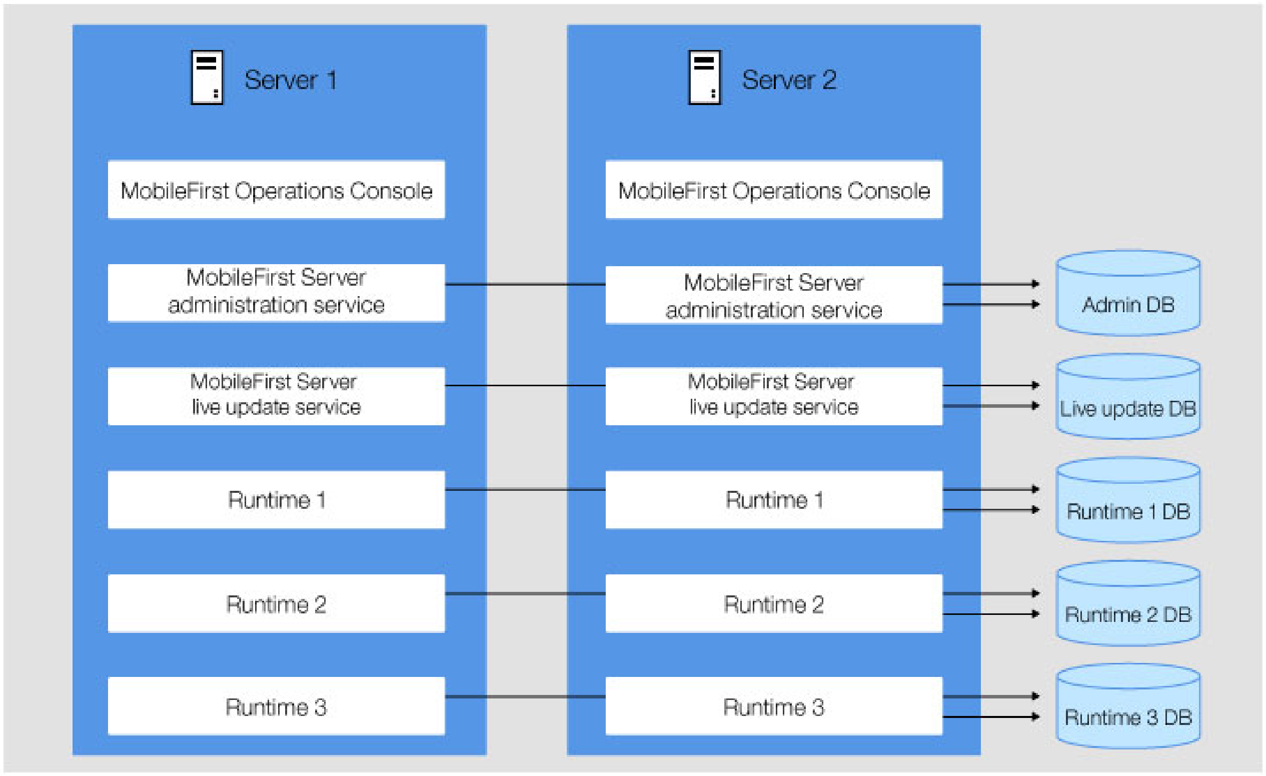

Server 1
Server 2
MobileFirst Operations Console
MobileFirst Server administration service
MobileFirst Server live update service
Runtime 1
Runtime 2
Runtime 3
MobileFirst Operations Console
MobileFirst Server administration service
MobileFirst Server live update service
Runtime 1
Runtime 2
Runtime 3
Admin DB
Live update DB
Runtime 1 DB
Runtime 2 DB
Runtime 3 DB

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 and one live update 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 live update service uses its own live update database schema. All deployed instances of the same live update service share the same live update 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 following tables list these properties. For instructions about how to install a server farm, see "Installing a server farm" on page 6-139. For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174 and "List of JNDI properties for MobileFirst runtime" on page 6-183.

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 the runtimes.

Table 6-14. Global JNDI properties for administration services and runtimes in server farm topology of WebSphere Application Server Liberty profile.

JNDI properties	Values
`mfp.topology.platform`	`Liberty`
`mfp.topology.clustermode`	`Farm`
`mfp.admin.jmx.host`	The host name of the WebSphere Application Server Liberty profile server
`mfp.admin.jmx.port`	The ort 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="*" />`

Table 6-14. Global JNDI properties for administration services and runtimes in server farm topology of WebSphere Application Server Liberty profile (continued).

JNDI properties	Values

mfp.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>MfpRESTUser</user>`
	`</administrator-role>`

mfp.admin.jmx.pwd	The password of the WebSphere Application Server Liberty administrator user.

The **mfp.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 **mfp.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **mfp.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 the runtimes.

Table 6-15. Global JNDI properties for administration services and runtimes in server farm topology of Apache Tomcat.

JNDI properties	Values
mfp.topology.platform	`Tomcat`
mfp.topology.clustermode	`Farm`

JVM properties are also required to define Java Management Extensions (JMX) Remote Method Invocation (RMI). For more information, see "Configuring JMX connection for Apache Tomcat" on page 6-101.

The **mfp.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 **mfp.admin.environmentid** JNDI property.

- On each runtime, the same value for the local **mfp.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 the runtimes.

Table 6-16. Global JNDI properties for administration services and runtimes in server farm topology of WebSphere Application Server full profile.

JNDI properties	Values
mfp.topology.platform	WAS
mfp.topology.clustermode	Farm
mfp.admin.jmx.connector	SOAP

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

JNDI properties	Values
mfp.admin.jmx.user	The user name of WebSphere Application Server. This user must be defined in the WebSphere Application Server user registry.
mfp.admin.jmx.pwd	The password of the WebSphere Application Server user.
mfp.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 **mfp.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **mfp.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

Liberty collective topology:

You can deploy the MobileFirst Server components in a Liberty collective topology.

In the Liberty collective topology, the MobileFirst Server administration components (MobileFirst Operations Console, the administration service, and the live update service) are deployed in a collective controller and the MobileFirst runtimes in collective member. This topology supports only asymmetric deployment, the runtimes cannot be deployed in a collective controller.

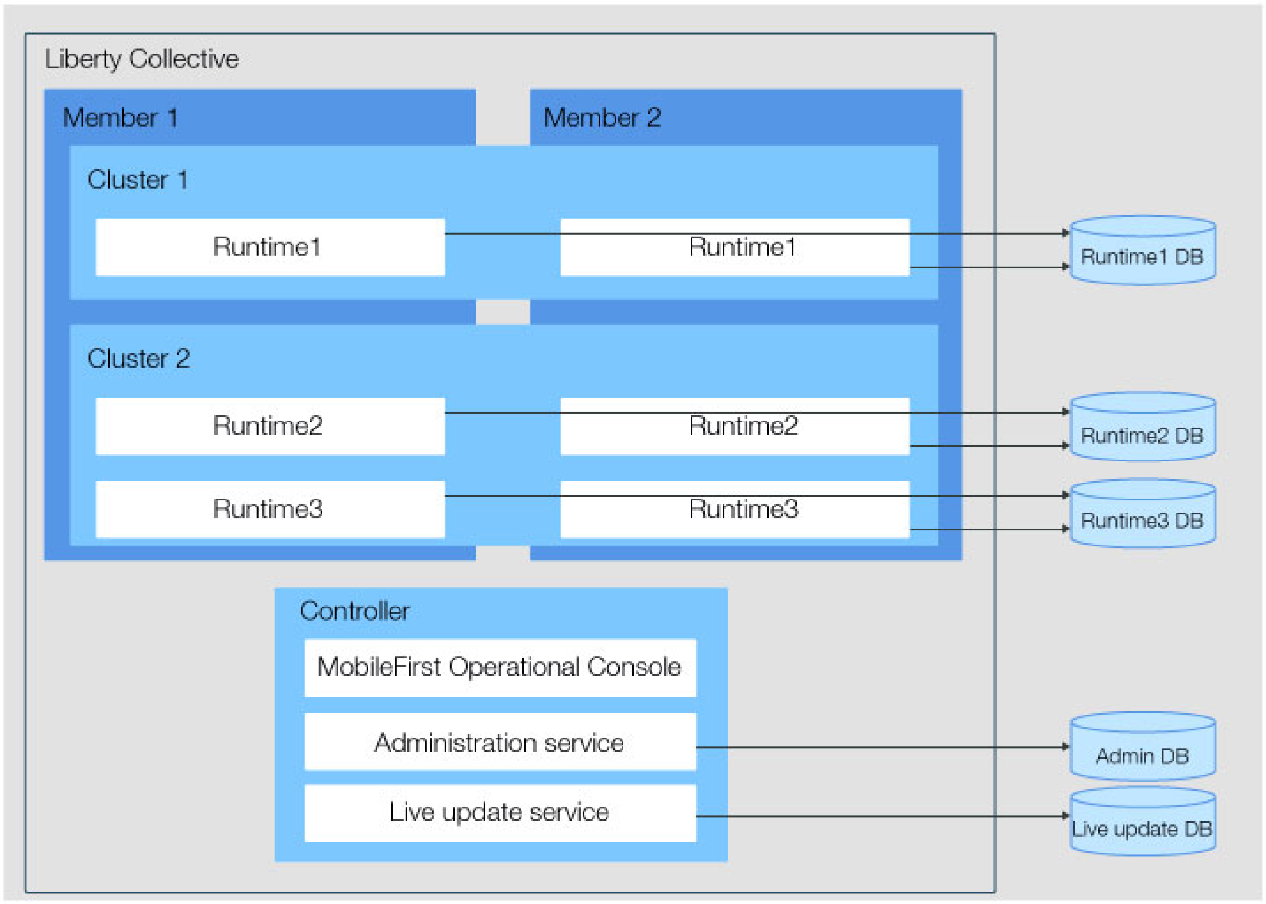


Figure 6-3. The topology of a Liberty collective

The deployment of this topology has the following characteristics:

- One or several administration components can be deployed in one or several controllers of the collective. Each instance of MobileFirst Operations Console communicates with one administration service and one live update service.
- One or several runtimes can be deployed in the cluster members of the collective.
- One MobileFirst Operations Console manages several runtimes that are deployed in the cluster members of the collective.
- One runtime is managed by only one MobileFirst Operations Console.
- Each administration service uses its own administration database schema.
- Each live update service uses its own live update database schema.
- Each runtime uses its own runtime database schema.

Configuration of JNDI properties

The following tables list the 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 more information about these properties, see “List of JNDI properties for MobileFirst Server administration service” on page 6-174 and “List of JNDI properties for MobileFirst runtime” on page 6-183 page 6-181. For instructions about how to install a Liberty collective manually, see "Manual installation on WebSphere Application Server Liberty collective" on page 6-121.

The following global JNDI properties are required for the administration services:

Table 6-17. Global JNDI properties for the administration services.

JNDI properties	Values
`mfp.topology.platform`	`Liberty`
`mfp.topology.clustermode`	`Cluster`
`mfp.admin.serverid`	`controller`
`mfp.admin.jmx.host`	The host name of the Liberty controller.
`mfp.admin.jmx.port`	The 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 Liberty controller. `<httpEndpoint id="defaultHttpEndpoint" httpPort="9080" httpsPort="9443" host="*"/>`
`mfp.admin.jmx.user`	The user name of the controller administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the Liberty controller. `<administrator-role> <user>MfpRESTUser</user> </administrator-role>`
`mfp.admin.jmx.pwd`	The password of the Liberty controller administrator user.

Several administration components can be deployed to enable the controller to run 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 **`mfp.admin.environmentid`** JNDI property.

The following global JNDI properties are required for the runtimes:

Table 6-18. Global JNDI properties for the runtimes.

JNDI properties	Values
`mfp.topology.platform`	`Liberty`
`mfp.topology.clustermode`	`Cluster`
`mfp.admin.serverid`	A value that identifies uniquely the collective member. It must be different for each member in the collective. The value `controller` cannot be used as it is reserved for the collective controller.
`mfp.admin.jmx.host`	The host name of the Liberty controller.

Table 6-18. Global JNDI properties for the runtimes. (continued)

JNDI properties	Values
mfp.admin.jmx.port	The 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 Liberty controller. `<httpEndpoint id="defaultHttpEndpoint" httpPort="9080" httpsPort="9443" host="*"/>`
mfp.admin.jmx.user	The user name of the controller administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the Liberty controller. `<administrator-role> <user>MfpRESTUser</user> </administrator-role>`
mfp.admin.jmx.pwd	The password of the Liberty controller administrator user.

The following JNDI property is required for the runtime when several controllers (replicas) using the same administration components are used:

Table 6-19. JNDI properties for the runtime.

JNDI properties	Values
mfp.admin.jmx.replica	Endpoint list of the different controller replicas with the following syntax: `replica-1 hostname:replica-1 port, replica-2 hostname:replica-2 port,..., replica-n hostname:replica-n port`

When several administration components are deployed in the controller, each runtime must have the same value for the local **mfp.admin.environmentid** JNDI property as the value that is 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 (MobileFirst Operations Console, the administration service, and the live update service) in one cluster and the runtimes managed by these components in another cluster.

Symmetric deployment in the same server or cluster

Figure 6-4 on page 6-95 shows symmetric deployment where the runtimes and the administration components are deployed in the same server or cluster.

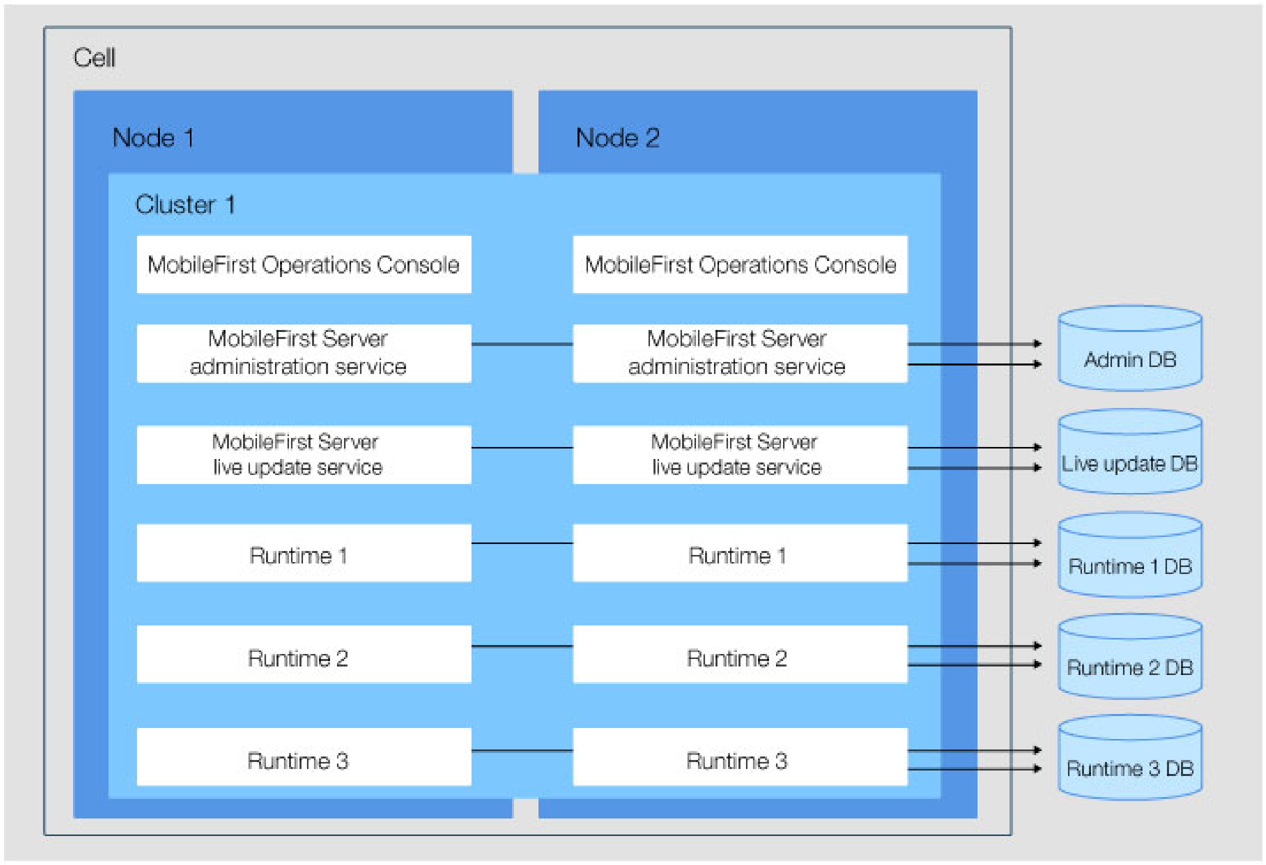

Cell
Node 1
Node 2
Cluster 1
MobileFirst Operations Console
MobileFirst Operations Console
MobileFirst Server administration service
MobileFirst Server administration service
MobileFirst Server live update service
MobileFirst Server live update service
Runtime 1
Runtime 1
Runtime 2
Runtime 2
Runtime 3
Runtime 3
Admin DB
Live update DB
Runtime 1 DB
Runtime 2 DB
Runtime 3 DB

Figure 6-4. 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 and one live update 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 live update service uses its own live update database schema.
- Each runtime uses its own runtime database schema.

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

Figure 6-5 on page 6-96 shows a topology where the runtimes are deployed in a different server or cluster from the administration services.

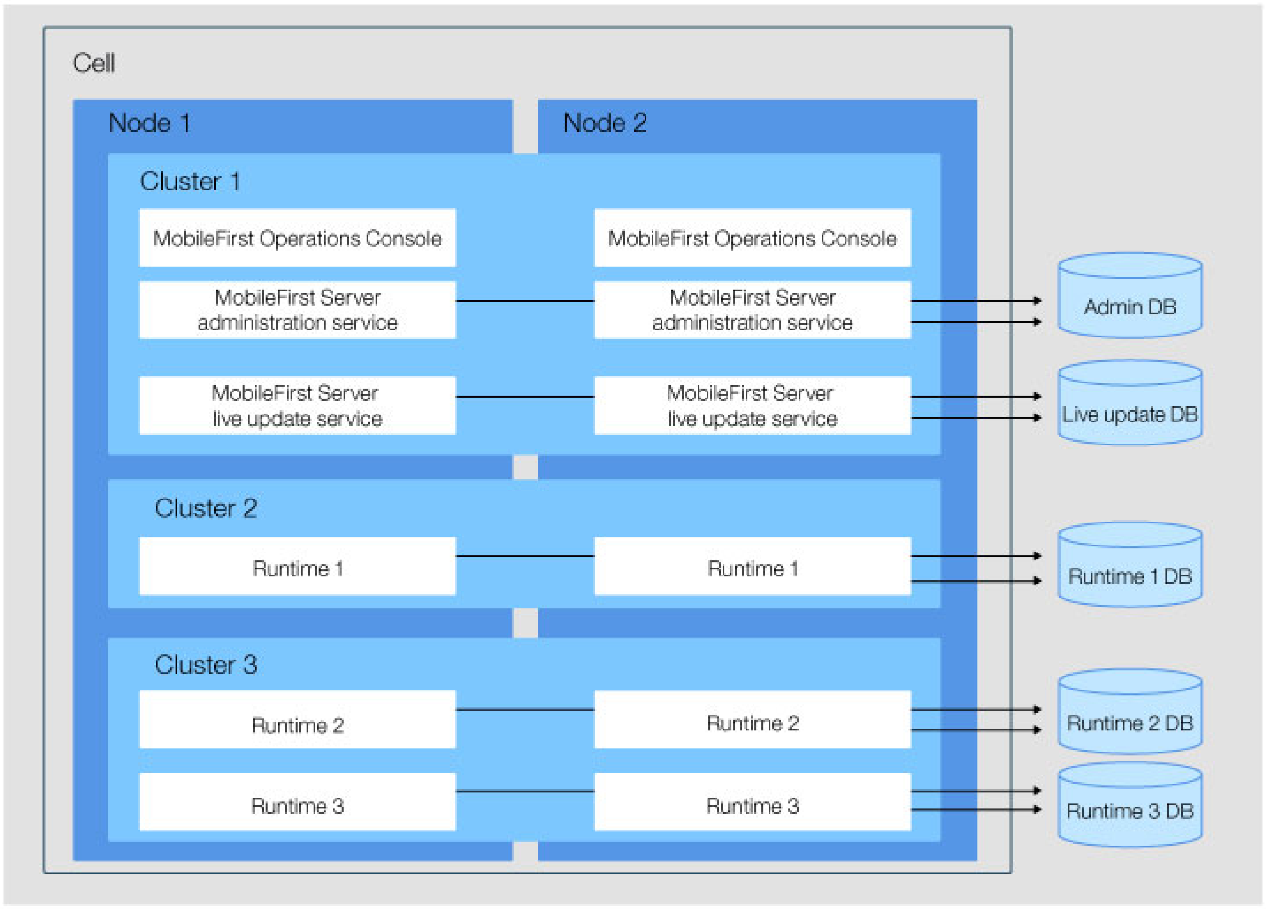

Cell
Node 1
Node 2
Cluster 1
MobileFirst Operations Console
MobileFirst Operations Console
MobileFirst Server administration service
MobileFirst Server administration service
MobileFirst Server live update service
MobileFirst Server live update service
Cluster 2
Runtime 1
Runtime 1
Cluster 3
Runtime 2
Runtime 2
Runtime 3
Runtime 3
Admin DB
Live update DB
Runtime 1 DB
Runtime 2 DB
Runtime 3 DB

Figure 6-5. 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 and one live update 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 live update service uses its own live update 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 6-6 on page 6-97 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.

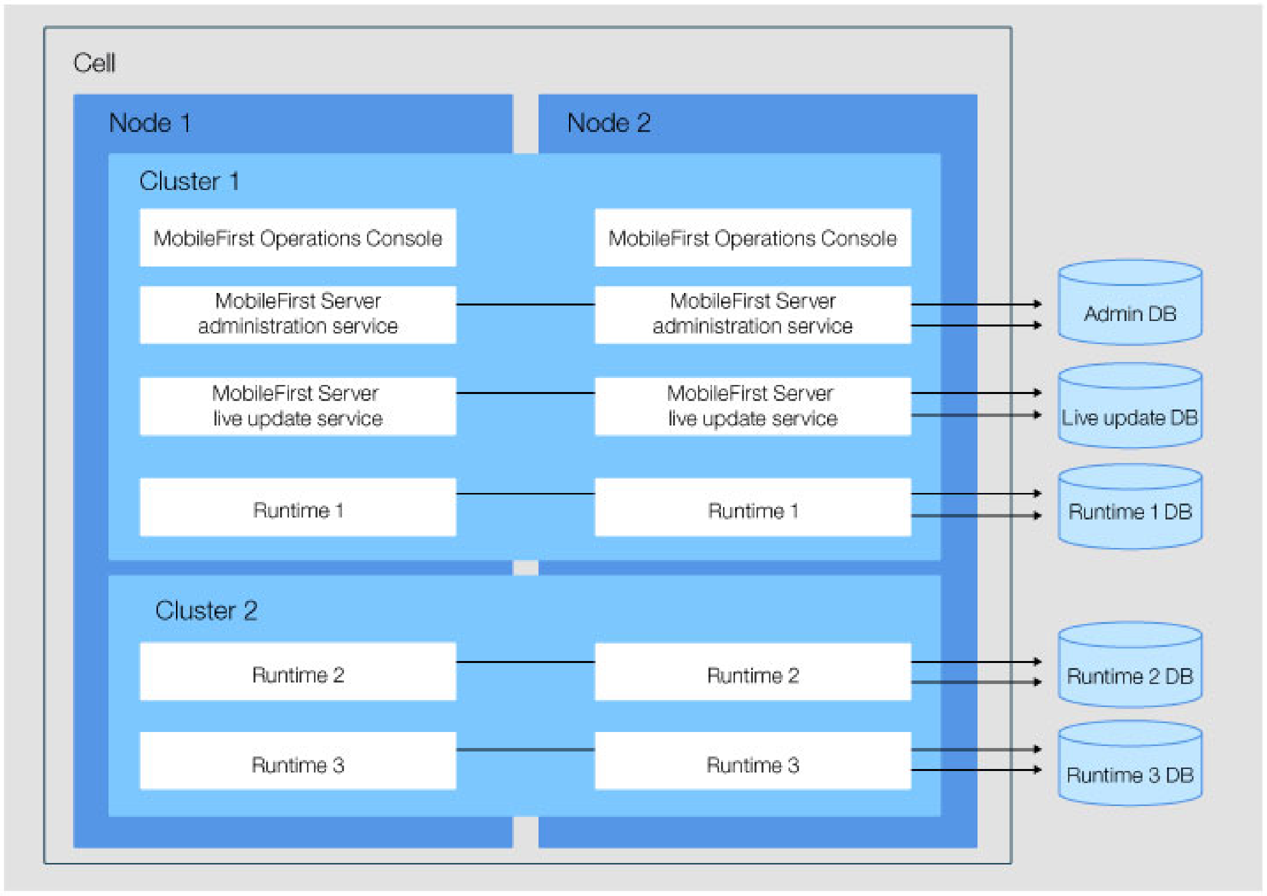

Cell
Node 1
Node 2
Cluster 1
MobileFirst Operations Console
MobileFirst Operations Console
MobileFirst Server administration service
MobileFirst Server administration service
MobileFirst Server live update service
MobileFirst Server live update service
Runtime 1
Runtime 1
Cluster 2
Runtime 2
Runtime 2
Runtime 3
Runtime 3
Admin DB
Live update DB
Runtime 1 DB
Runtime 2 DB
Runtime 3 DB

Figure 6-6. 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 and one live update 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 live update service uses its own live update 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 service" on page 6-174 and "List of JNDI properties for MobileFirst runtime" on page 6-183

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

Table 6-20. Local JNDI properties for administration services and runtimes in WebSphere Application Server Network Deployment topologies.

JNDI properties	Values
mfp.topology.platform	WAS
mfp.topology.clustermode	Cluster
mfp.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. You must use RMI if the SOAP port is disabled.
mfp.admin.jmx.dmgr.host	The host name of the deployment manager.
mfp.admin.jmx.dmgr.port	The RMI or the SOAP port used by the deployment manager, depending on the value of **mfp.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.

When several administration components are deployed, you must specify:
- On each administration service, a unique value for the local **mfp.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **mfp.admin.environmentid** as the value defined for the administration service that manages that runtime.

If the virtual host that is mapped to an administration service application is not the default host, you must set the following properties on the administration service application:
- **mfp.admin.jmx.user**: the user name of the WebSphere Application Server administrator
- **mfp.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 service.

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 6-21. JNDI properties for reverse proxy

JNDI properties	Values
mfp.admin.proxy.protocol	The protocol that is used to communicate with the reverse proxy. It can be HTTP or HTTPS.
mfp.admin.proxy.host	The host name of the reverse proxy.
mfp.admin.proxy.port	The port number of the reverse proxy.

The **mfp.admin.endpoint** property that references the URL of the reverse proxy is also required for MobileFirst Operations Console.

Constraints on MobileFirst Server push service:

Find out the constraints of deploying push service application.

The push service can be on the same application server as the administration service or the runtime, or can be on a different application server. The URL used by the client apps to contact the push service is the same as the URL used by the client apps to contact the runtime, excepted that the context root of the runtime is replaced by imfpush. If you install the push service on a different server than the runtime, your HTTP server must direct the traffic to the /imfpush context root to a server where the push service runs.

For more information about the JNDI properties that are needed to adapt the installation to a topology, see "MobileFirst Server administration service to MobileFirst Server push service, and to the authorization server " on page 6-83. The push service must be installed with the context root /imfpush.

Multiple MobileFirst runtimes:

Find out the details about deploying multiple runtimes on the same server.

You can install multiple runtimes. Each runtime must have its own context root, and all of these runtimes are managed by the same MobileFirst Server administration service and MobileFirst Operations Console.

The constraints as described in "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84 applies. Each runtime (with its context root) must have its own database tables.

Multiple instances of MobileFirst Server on the same server or WebSphere Application Server cell:

By defining a common environment ID, multiple instances of MobileFirst Server are possible to be installed on the same server.

You can install multiple instances of MobileFirst Server administration service, MobileFirst Server live update service, and MobileFirst runtime on the same application server or WebSphere Application Server cell. However, you must distinguish their installations with the JNDI variable: **`mfp.admin.environmentid`**, which is a variable of the administration service and of the runtime. The administration service manages only the runtimes that have the same environment identifier. As such, only the runtime components and the administration service that have the same value for **`mfp.admin.environmentid`** are considered as part of the same installation.

Installing MobileFirst Server to an application server

The installation of the components can be done by using Ant Tasks, the Server Configuration Tool, or manually. Find out the prerequisite and the details about the installation process so that you can install the components on the application server successfully.

For an overview of the installation process, see "Tutorials about MobileFirst Server installation" on page 6-4.

Before you proceed with installing the components to the application server, ensure that the databases and the tables for the components are prepared and ready to use. For more information, see "Setting up databases" on page 6-63.

The server topology to install the components must also be defined. See "Topologies and network flows" on page 6-78.

You can install the components with the following methods:
- Ant tasks
- Server Configuration Tool
- Manual installation

You can find the information about installing the MobileFirst Server components to one or more application servers in the following topics.

Application server prerequisites:

Depending on your choice of the application server, select one of the following topics to find out the prerequisites that you must fulfill before you install the MobileFirst Server components.

Apache Tomcat prerequisites:

MobileFirst Server has some requirements for the configuration of Apache Tomcat that are detailed in the following topics.

Ensure that you fulfill the following criteria:
- Use a supported version of Apache Tomcat. See "System requirements" on page 2-7.
- Apache Tomcat must be run with JRE 7.0 or later.

- The JMX configuration must be enabled to allow the communication between the administration service and the runtime component. The communication uses RMI as described in "Configuring JMX connection for Apache Tomcat."

Configuring JMX connection for 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 *tomcat_install_dir*/bin/setenv.bat and *tomcat_install_dir*/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 *tomcat_install_dir*/bin/startup.bat, or *tomcat_install_dir*/bin/catalina.bat.
- The setenv.sh file is used if you start Apache Tomcat with <*tomcatInstallDir*>/bin/startup.sh, or *tomcat_install_dir*/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.

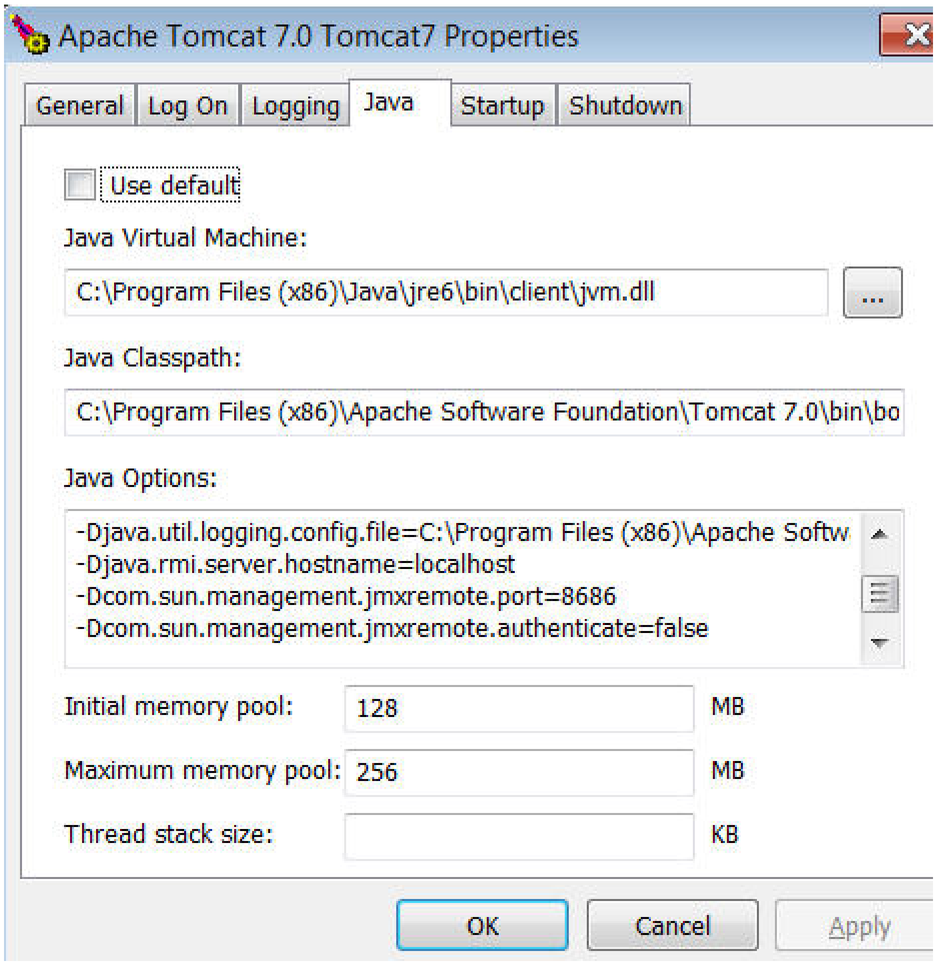
   The following environment properties must also be added to the Context section of the administration service application in the `server.xml` file, such as in the following example:

```
<Context docBase="mfpadmin" path="/mfpadmin ">
    <Environment name="mfp.admin.rmi.registryPort" value="registryPort" type="java.lang.String" override="false"/>
    <Environment name="mfp.admin.rmi.serverPort" value="serverPort" type="java.lang.String" override="false"/>
</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 installed Apache Tomcat with the Apache Tomcat Windows Service Installer instead of adding the options to `CATALINA_OPTS`, run *`tomcat_install_dir`*`/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

WebSphere Application Server Liberty prerequisites:

IBM MobileFirst Platform Server has some requirements for the configuration of the Liberty server that are detailed in the following topics.

Ensure that you fulfill the following criteria:

- Use a supported version of Liberty. See "System requirements" on page 2-7.
- Liberty must be run with JRE 7.0 or later. JRE 6.0 is not supported.
- Some versions of Liberty support both the features of Java EE 6 and Java EE 7. For example, `jdbc-4.0` Liberty feature is part of Java EE 6, whereas `jdbc-4.1` Liberty feature is part of Java EE 7. MobileFirst Server V8.0.0 can be installed with Java EE 6 or Java EE 7 features. However, if you want to run an older version of MobileFirst Server on the same Liberty server, you must use the Java EE 6 features. MobileFirst Server V7.1.0 and earlier, does not support the Java EE 7 features.
- JMX must be configured as documented in "Configuring JMX connection for WebSphere Application Server Liberty profile."
- For an installation in a production environment, you might want to start the Liberty server as a service on Windows, Linux, or UNIX systems so that:
  - The MobileFirst Server components are started automatically when the computer starts.
  - The process that runs Liberty server is not stopped when the user, who started the process, logs out.
- MobileFirst Server V8.0.0 cannot be deployed in a Liberty server that contains the deployed MobileFirst Server components from the previous versions.
- For an installation in a Liberty collective environment, the Liberty collective controller and the Liberty collective cluster members must be configured as documented in Configuring a Liberty collective.

Configuring JMX connection for WebSphere Application Server Liberty profile:

You must configure a secure JMX connection for 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 as described in 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:

  ```
  liberty_install_dir/bin/productInfo featureInfo
  ```

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

WebSphere Application Server and WebSphere Application Server Network Deployment prerequisites:

IBM MobileFirst Platform Server has some requirements for the configuration of WebSphere Application Server and WebSphere Application Server Network Deployment that are detailed in the following topics.

Ensure that you fulfill the following criteria:
- Use a supported version of WebSphere Application Server. See "System requirements" on page 2-7.
- The application server must be run with JRE 7.0. By default, WebSphere Application Server uses Java 6.0 SDK. To switch to Java 7.0 SDK, see Switching to Java 7.0 SDK in WebSphere Application Server.
- The administrative security must be turned on. MobileFirst Operations Console, the MobileFirst Server administration service, and the MobileFirst Server configuration service are protected by security roles. For more information, see Enabling security.
- The JMX configuration must be enabled to allow the communication between the administration service and the runtime component. The communication uses SOAP. For WebSphere Application Server Network Deployment, RMI can be used. For more information, see "Configuring JMX connection for WebSphere Application Server and WebSphere Application Server Network Deployment."

Configuring JMX connection for 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
- MobileFirst Server 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. MobileFirst Server uses the SOAP port by default. If both the SOAP and RMI ports are deactivated, MobileFirst Server does not run.
- RMI is only supported by WebSphere Application Server Network Deployment. RMI is not supported by a stand-alone profile, or a WebSphere Application Server server farm.
- You must activate Administrative and Application Security.

File system prerequisites:

To install IBM MobileFirst Platform Server 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 MobileFirst Server

For WebSphere Application Server Liberty profile, you must have the required permission 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 required permission 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 required permission 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.

Installing with the Server Configuration Tool:

Use the Server Configuration Tool to install the MobileFirst Server components to your application server.

The Server Configuration Tool can set up the database and install the components to an application server. This tool is meant for a single user. The configuration files are store on the disk. The directory where they are stored can be modified with menu **File** > **Preferences**. The files must be used only by one instance of the Server Configuration Tool at the time. The tool does not manage concurrent access to the same file. If you have multiple instances of the tool accessing the same file, the data might be lost. For more information about how the tool creates and setup the databases, see “Create the database tables with the Server Configuration Tool” on page 6-70. If the databases exist, the tool can detect them by testing the presence and the content of some test tables and does not modify these database tables.

Supported operating systems:

Find out the operating systems that are supported by the Server Configuration Tool.

You can use the Server Configuration Tool if you are on the following operating systems:

- Windows x86 or x86-64
- Mac OS x86-64
- Linux x86 or Linux x86-64

The tool is not available on other operating systems. You need to use Ant tasks to install the MobileFirst Server components as described in "Installing with Ant Tasks" on page 6-110.

Supported topologies:

Find out the topologies that are supported by the Server Configuration Tool to install the MobileFirst Server components.

The Server Configuration Tool installs the MobileFirst Server components with the following topologies:

- All components (MobileFirst Operations Console, the MobileFirst Server administration service, the MobileFirst Server live update service, and the MobileFirst runtime) are in the same application server. However, on WebSphere Application Server Network Deployment when you install on a cluster, you can specify a different cluster for the administration and live update services, and for the runtime. On Liberty collective, MobileFirst Operations Console, the administration service, and the live update service are installed in a collective controller and the runtime in a collective member.
- If the MobileFirst Server push service is installed, it is also installed on the same server. However, on WebSphere Application Server Network Deployment when you install on a cluster, you can specify a different cluster for the push service. On Liberty collective, the push service is installed in a Liberty member that can be the same as the one where the runtime is installed.
- All the components use the same database system and the user. For DB2, all the components also use the same schema.
- The Server Configuration Tool installs the components for a single server except for Liberty collective and WebSphere Application Server Network Deployment for asymmetric deployment. For an installation on multiple servers, a farm must be configured after the tool is run. The server farm configuration is not required on WebSphere Application Server Network Deployment.

For other topologies or other database settings, you can install the components with Ant Tasks or manually instead.

Running the Server Configuration Tool:

Follow the instructions to run the Server Configuration Tool and install MobileFirst Server on the application server.

Before you begin

Before you run the Server Configuration Tool, make sure that the following requirements are fulfilled:

- The databases and the tables for the components are prepared and ready to use. See "Setting up databases" on page 6-63.
- The server topology to install the components is decided. See "Topologies and network flows" on page 6-78.

- The application server is configured. See “Application server prerequisites” on page 6-100.
- The user that runs the tool has the specific file system privileges. See “File system prerequisites” on page 6-104.

Procedure

1. Start the Server Configuration Tool.
   - On Linux, from application shortcuts **Applications** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Windows, click **Start** > **Programs** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Mac OS, open a shell console. Go to `*mfp_server_install_dir*/shortcuts` and type `./configuration-tool.sh`.

     The *mfp_server_install_dir* directory is where you installed MobileFirst Server.
2. Select **File** > **New Configuration** to create a MobileFirst Server Configuration.
   a. In the **Configuration Details** panel, enter the context root of the administration service and the runtime component. You might want to enter an environment ID.

      An environment ID is used in advanced use cases, for example when multiple installations of MobileFirst Server are made on the same application server or same WebSphere Application Server cell. See “Multiple instances of MobileFirst Server on the same server or WebSphere Application Server cell” on page 6-100.

   b. In the **Console Settings** panel, select whether to install MobileFirst Operations Console or not.

      If the console is not installed, you need to use command line tools (**`mfpdev`** or **`mfpadm`**) or the REST API to interact with the MobileFirst Server administration service.

   c. In the **Database Selection** panel, select the database management system that you plan to use.

      All the components use the same database type and the same database instance. For more information about the database panes, see “Create the database tables with the Server Configuration Tool” on page 6-70.

   d. In the **Application Server Selection** panel, select the type of application server where you want to deploy MobileFirst Server.
3. In the **Application Server Settings** panel, choose the application server and do the following steps:
   - For an installation on WebSphere Application Server Liberty:
     – Enter the installation directory of Liberty and the name of the server where you want to install MobileFirst Server.

     – You can create a default user to log in the console. This user is created in the Liberty Basic registry. For a production installation, you might want to clear the `Create a default user` option and to configure the user access after the installation. For more information, see “Configuring user authentication for MobileFirst Server administration” on page 6-166.

     – Select the deployment type: `Standalone deployment` (default), `Server farm deployment`, or `Liberty collective deployment`.

   If the `Liberty collective deployment` option is selected, do the following steps:

   a. Specify the Liberty collective server:

    - Where the administration service, MobileFirst Operations Console and the live update service are installed. The server must be a Liberty collective controller.
    - Where the runtime is installed. The server must be a Liberty collective member.
    - Where the push service is installed. The server must be a Liberty collective member.
  b. Enter the server ID of the member. This identifier must be different for each member in the collective.
  c. Enter the cluster name of the collective members.
  d. Enter the controller host name and HTTPS port number. The values must be the same as the one that is defined in the `<variable>` element inside the `server.xml` file of the Liberty collective controller.
  e. Enter the controller administrator user name and password.
- For an installation on WebSphere Application Server or WebSphere Application Server Network Deployment:
  - Enter the installation directory of WebSphere Application Server.
  - Select the WebSphere Application Server profile where you want to install MobileFirst Server. If you install on WebSphere Application Server Network Deployment, select the profile of the deployment manager. On the deployment manager profile, you can select a scope (`Server` or `Cluster`). If you select `Cluster`, you must specify the cluster:
    - Where the runtime is installed.
    - Where the administration service, MobileFirst Operations Console and the live update service are installed.
    - Where the push service is installed.
  - Enter an administrator login ID and password. The administrator user must have an administrator role.
  - If you select the `Declare the WebSphere Administrator as an administrator user in IBM MobileFirst Platform Operations Console` option, then the user that is used to install MobileFirst Server is mapped to the administration security role of the console and can log in to the console with administrator privileges. This user is also mapped to the security role of the live update service. The user name and password are set as JNDI properties (**`mfp.config.service.user`** and **`mfp.config.service.password`**) of the administration service.
  - If you do not select the `Declare the WebSphere Administrator as an administrator user in IBM MobileFirst Platform Operations Console` option, then before you can use MobileFirst Server, you must do the following tasks:
    - Enable the communication between the administration service and the live update service by:
      - Mapping a user to the security role `configadmin` of the live update service.
      - Adding the login ID and password of this user in the JNDI properties (**`mfp.config.service.user`** and **`mfp.config.service.password`**) of the administration service.
      - Map one or more users to the security roles of the administration service and MobileFirst Operations Console. See "Configuring user authentication for MobileFirst Server administration" on page 6-166.
- For an installation on Apache Tomcat:

- Enter the installation directory of Apache Tomcat.
- Enter the port that is used for the JMX communication with RMI. By default, the value is 8686. The Server Configuration Tool modifies the `*tomcat_install_dir*/bin/setenv.bat` or `*tomcat_install_dir*/bin/setenv.sh` file to open this port. If you want to open the port manually, or have already some code that opens the port in `setenv.bat` or `setenv.sh`, do not use the tool. Install with Ant tasks instead. An option to open the RMI port manually is provided for an installation with Ant tasks.
- Create a default user to log in the console. This user is also created in the `tomcat-users.xml` configuration file. For a production installation, you might want to clear the `Create a default user` option and to configure the user access after the installation. For more information, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

4. In the **Push Service Settings** panel, select the `Install the Push service` option if you want the push service to be installed in the application server. The context root is `imfpush`. To enable the communication between the push service and the administration service, you need to define the following parameters:
   a. Enter the URL of the push service and the URL of the runtime. This URL can be computed automatically if you install on Liberty, Apache Tomcat, or stand-alone WebSphere Application Server. It uses the URL of the component (the runtime or the push service) on the local server. If you install on WebSphere Application Server Network Deployment or the communications go through a web proxy or load balancer, you must enter the URL manually.
   b. Enter the confidential client IDs and secret for the OAuth communication between the services. Otherwise, the tool generates default values and random passwords.
5. In the **Analytics Settings** panel, select the `Enable the connection to the Analytics server` if MobileFirst Analytics is installed. Enter the following connection settings:
   - The URL of the Analytics console.
   - The URL of the Analytics server (the Analytics data service).
   - The user login ID and password that is allowed to publish data to the Analytics server.

   The tool configures the runtime and the push service to send data to the Analytics server.
6. Click **Deploy** to proceed with the installation.

What to do next

After the installation is completed successfully, restart the application server in the case of Apache Tomcat or Liberty profile.

If Apache Tomcat is launched as a service, the `setenv.bat` or `setenv.sh` file that contains the statement to open the RMI might not be read. As a result, MobileFirst Server might not be able to work correctly. To set the required variables, see "Configuring JMX connection for Apache Tomcat" on page 6-101.

On WebSphere Application Server Network Deployment, the applications are installed but not started. You need to start them manually. You can do that from the WebSphere Application Server administration console.

Keep the configuration file in the Server Configuration Tool. You might reuse it to install the interim fixes. The menu to apply an interim fix is **Configurations** > **Replace the deployed WAR files**.

Applying a fix pack by using the Server Configuration Tool:

You can apply a fix pack or an interim fix by using the Server Configuration Tool if MobileFirst Server is installed previously with the tool.

About this task

If MobileFirst Server is installed with the tool and the configuration file is kept, you can apply a fix pack or an interim fix by reusing the configuration file.

Procedure

1. Start the Server Configuration Tool.
   - On Linux, from application shortcuts **Applications** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Windows, click **Start** > **Programs** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Mac OS, open a shell console. Go to `*mfp_server_install_dir*/shortcuts` and type `./configuration-tool.sh`.

     The *mfp_server_install_dir* directory is where you installed MobileFirst Server.
2. Click **Configurations** > **Replace the deployed WAR files** and select an existing configuration to apply the fix pack or an interim fix.

Installing with Ant Tasks:

Use Ant tasks to install the MobileFirst Server components to your application server.

You can find the sample configuration files for installing MobileFirst Server in the *mfp_install_dir*`/MobileFirstServer/configuration-samples` directory.

You can also create a configuration with the Server Configuration Tool and export the Ant files by using **File** > **Export Configuration as Ant Files...**. The sample Ant files have the same limitations as the Server Configuration Tool:

- All components (MobileFirst Operations Console, MobileFirst Server administration service, MobileFirst Server live update service, the MobileFirst Server artifacts, and MobileFirst runtime) are in the same application server. However, on WebSphere Application Server Network Deployment when you install on a cluster, you can specify a different cluster for the administration and live update services, and for the runtime.
- If the MobileFirst Server push service is installed, it is also installed on the same server. However, on WebSphere Application Server Network Deployment when you install on a cluster, you can specify a different cluster for the push service.
- All the components use the same database system and the user. For DB2, all the components also use the same schema.
- The Server Configuration Tool installs the components for a single server. For an installation on multiple servers, a farm must be configured after the tool is run. The server farm configuration is not supported on WebSphere Application Server Network Deployment.

You can configure the MobileFirst Server services to run in server farm with Ant tasks. To include your server in a farm, you need to specify some specific attributes that configure your application server accordingly. For more information about configuring a server farm with Ant tasks, see "Installing a server farm with Ant tasks" on page 6-141.

For other topologies that are supported in "Topologies and network flows" on page 6-78, you can modify the sample Ant files.

The references to the Ant tasks are as follows:
- "Ant tasks for installation of MobileFirst Operations Console, MobileFirst Server artifacts, MobileFirst Server administration, and live update services" on page 6-274
- "Ant tasks for installation of MobileFirst Server push service" on page 6-287
- "Ant tasks for installation of MobileFirst runtime environments" on page 6-293

For an overview of installing with the sample configuration file and tasks, see "Installing MobileFirst Server in command line mode" on page 6-22.

You can run an Ant file with the Ant distribution that is part of the product installation. For example, if you have WebSphere Application Server Network Deployment cluster and your database is IBM DB2, you can use the `mfp_install_dir/MobileFirstServer/configuration-samples/configure-wasnd-cluster-db2.xml` Ant file. After you edit the file and enter all the required properties, you can run the following commands from `mfp_install_dir/MobileFirstServer/configuration-samples` directory:
- `mfp_install_dir/shortcuts/ant -f configure-wasnd-cluster-db2.xml help` - This command displays the list of all the possible targets of the Ant file, to install, uninstall, or update some components.
- `mfp_install_dir/shortcuts/ant -f configure-wasnd-cluster-db2.xml install` - This command installs MobileFirst Server on the WebSphere Application Server Network Deployment cluster, with DB2 as a data source by using the parameters that you entered in the properties of the Ant file.

After the installation, make a copy of the Ant file so that you can reuse it to apply a fix pack. For more information, see "Applying a fix pack by using the Ant files."

Applying a fix pack by using the Ant files:

You can apply a fix pack with Ant tasks if MobileFirst Server is installed with Ant tasks.

Updating with the sample Ant file:
About this task

If you use the sample Ant files that are provided in the `mfp_install_dir/MobileFirstServer/configuration-samples` directory to install MobileFirst Server, you can reuse a copy of this Ant file to apply a fix pack. For password values, you can enter ************ (12 stars) instead of the actual value, to be prompted interactively when the Ant file is run.

To apply a fix pack, do the following steps.

Procedure

1. Verify the value of the **mfp.server.install.dir** property in the Ant file. It must point to the directory that contains the product with the fix pack applied. This value is used to take the updated MobileFirst Server WAR files.
2. Run the command:

   ```
   mfp_install_dir/shortcuts/ant -f your_ant_file update
   ```

Updating with own Ant file:
About this task

If you use your own Ant file, make sure that for each installation task (**installmobilefirstadmin**, **installmobilefirstruntime**, and **installmobilefirstpush**), you have a corresponding update task in your Ant file with the same parameters. The corresponding update tasks are **updatemobilefirstadmin**, **updatemobilefirstruntime**, and **updatemobilefirstpush**.

To apply a fix pack with your own Ant file, do the following steps.

Procedure

1. Verify the class path of the `<taskdef>` element for the `mfp-ant-deployer.jar` file. It must point to the `mfp-ant-deployer.jar` file in an MobileFirst Server installation that the fix pack is applied. By default, the updated MobileFirst Server WAR files are taken from the location of `mfp-ant-deployer.jar`.
2. Run the update tasks (**updatemobilefirstadmin**, **updatemobilefirstruntime**, and **updatemobilefirstpush**) of your Ant file.

Sample Ant files modifications:

You can modify the sample Ant files that are provided in the *mfp_install_dir*/`MobileFirstServer/configuration-samples` directory to adapt to your installation requirements.

The following sections provide the details on how you can modify the sample Ant files to adapt the installation to your needs:

1. "Specify extra JNDI properties"
2. "Specify existing users" on page 6-113
3. "Specify Liberty Java EE level" on page 6-113
4. "Specify data source JDBC properties" on page 6-114
5. "Run the Ant files on a computer where MobileFirst Server is not installed" on page 6-114
6. "Specify WebSphere Application Server Network Deployment targets" on page 6-114
7. "Manual configuration of the RMI port on Apache Tomcat" on page 6-115

Specify extra JNDI properties

The **installmobilefirstadmin**, **installmobilefirstruntime**, and **installmobilefirstpush** Ant tasks declare the values for the JNDI properties that are required for the components to function. These JNDI properties are used to define the JMX communication, and also the links to other components (such the live update service, the push service, the analytics service, or the authorization server). However, you can also define values for other JNDI properties. Use the `<property>` element that exists for these three tasks. For a list of JNDI properties, see:

- "List of JNDI properties for MobileFirst Server administration service" on page 6-174
- "List of JNDI properties for MobileFirst Server push service" on page 6-186
- "List of JNDI properties for MobileFirst runtime" on page 6-183

For example:

```
<installmobilefirstadmin ..>
    <property name="mfp.admin.actions.prepareTimeout" value="3000000"/>
</installmobilefirstadmin>
```

Specify existing users

By default, the **installmobilefirstadmin** Ant task creates users:

- On WebSphere Application Server Liberty to define a Liberty administrator for the JMX communication.
- On any application server, to define a user that is used for the communication with the live update service.

To use an existing user instead of creating new user, you can do the following operations:

1. In the `<jmx>` element, specify a user and password, and set the value of the **createLibertyAdmin** attribute to `false`. For example:

```
<installmobilefirstadmin ...>
    <jmx libertyAdminUser="myUser" libertyAdminPassword="password" createLibertyAdmin="false" />
    ...
```

2. In the `<configuration>` element, specify a user and password and set the value of the **createConfigAdminUser** attribute to `false`. For example:

```
<installmobilefirstadmin ...>
    <configuration configAdminUser="myUser" configAdminPassword="password" createConfigAdminUser="false" />
    ...
```

Also, the user that is created by the sample Ant files is mapped to the security roles of the administration service and the console. With this setting, you can use this user to log on to MobileFirst Server after the installation. To change that behavior, remove the `<user>` element from the sample Ant files. Alternatively, you can remove the **password** attribute from the `<user>` element, and the user is not created in the local registry of the application server.

Specify Liberty Java EE level

Some distributions of WebSphere Application Server Liberty support features from Java EE 6 or from Java EE 7. By default, the Ant tasks automatically detect the features to install. For example, `jdbc-4.0` Liberty feature is installed for Java EE 6 and `jdbc-4.1` feature is installed in case of Java EE 7. If the Liberty installation supports both features from Java EE 6 and Java EE 7, you might want to force a certain level of features. An example might be that you plan to run both MobileFirst Server V8.0.0 and V7.1.0 on the same Liberty server. MobileFirst ServerV7.1.0 or earlier supports only Java EE 6 features.

To force a certain level of Java EE 6 features, use the **jeeversion** attribute of the `<websphereapplicationserver>` element. For example:

```
<installmobilefirstadmin execute="${mfp.process.admin}" contextroot="${mfp.admin.contextroot}">
    [...]
    <applicationserver>
      <websphereapplicationserver installdir="${appserver.was.installdir}"
        profile="Liberty"
        jeeversion="6">
```

Specify data source JDBC properties

You can specify the properties for the JDBC connection. Use the `<property>` element of a `<database>` element. The element is available in **`configureDatabase`**, **`installmobilefirstadmin`**, **`installmobilefirstruntime`**, and **`installmobilefirstpush`** Ant tasks. For example:

```
<configuredatabase kind="MobileFirstAdmin">
  <db2 database="${database.db2.mfpadmin.dbname}"
    server="${database.db2.host}"
    instance="${database.db2.instance}"
    user="${database.db2.mfpadmin.username}"
    port= "${database.db2.port}"
    schema = "${database.db2.mfpadmin.schema}"
    password="${database.db2.mfpadmin.password}">
   <property name="commandTimeout" value="10"/>
  </db2>
```

Run the Ant files on a computer where MobileFirst Server is not installed

To run the Ant tasks on a computer where MobileFirst Server is not installed, you need the following items:

- An Ant installation
- A copy of the `mfp-ant-deployer.jar` file to the remote computer. This library contains the definition of the Ant tasks.
- To specify the resources to be installed. By default, the WAR files are taken near the `mfp-ant-deployer.jar`, but you can specify the location of these WAR files. For example:

```
<installmobilefirstadmin execute="true" contextroot="/mfpadmin" serviceWAR="/usr/mfp/mfp-admin-service.war">
  <console install="true" warFile="/usr/mfp/mfp-admin-ui.war"/>
```

  For more information, see the Ant tasks to install each MobileFirst Server component at “Installation reference” on page 6-268.

Specify WebSphere Application Server Network Deployment targets

To install on WebSphere Application Server Network Deployment, the specified WebSphere Application Server profile must be the deployment manager. You can deploy on the following configurations:

- A cluster
- A single server
- A cell (all the servers of a cell)
- A node (all the servers of a node)

The sample files such as `configure-wasnd-cluster-<dbms>.xml`, `configure-wasnd-server-<dbms>.xml`, and `configure-wasnd-node-<dbms>.xml` contain the declaration to deploy on each type of target. For more information, see the Ant tasks to install each MobileFirst Server component at “Installation reference” on page 6-268.

Note: As of V8.0.0, the sample configuration file for the WebSphere Application Server Network Deployment cell is not provided.

Manual configuration of the RMI port on Apache Tomcat

By default, the Ant tasks modify the `setenv.bat` file or the `setenv.sh` file to open the RMI port. If you prefer to open the RMI port manually, add the **`tomcatSetEnvConfig`** attribute with the value as `false` to the `<jmx>` element of the **`installmobilefirstadmin`**, **`updatemobilefirstadmin`**, and **`uninstallmobilefirstadmin`** tasks.

Installing the MobileFirst Server components manually:

You can also install the MobileFirst Server components to your application server manually.

The following topics provide you the complete information to guide you through the installing process of the components on the supported applications in production.

Manual installation on WebSphere Application Server Liberty:

Find out more details on how to install the MobileFirst Server components on WebSphere Application Server Liberty.

For an overview of an installation of MobileFirst Server on Liberty profile, see "Tutorials about MobileFirst Server installation" on page 6-4.

Make sure that you have also fulfilled the requirements as documented in "WebSphere Application Server Liberty prerequisites" on page 6-103.

Topology constraints

The MobileFirst Server administration service, the MobileFirst Server live update service, and the MobileFirst runtime must be installed on the same application server. The context root of the live update service must be defined as `<`*`adminContextRoot`*`>config`. The context root of the push service must be `imfpush`. For more information about the constraints, see "Constraints on the MobileFirst Server components and MobileFirst Analytics" on page 6-84.

Application server settings

You must configure the `<webContainer>` element to load the servlets immediately. This setting is required for the initialization through JMX. For example:

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

Optionally, to avoid timeout issues that break the startup sequence of the runtime and the administration service on some Liberty versions, change the default `<executor>` element. Set large values to the **`coreThreads`** and **`maxThreads`** attributes. For example:

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

You might also configure the `<tcpOptions>` element and set the **`soReuseAddr`** attribute to `true`.

```
<tcpOptions soReuseAddr="true"/>
```

Liberty features required by the MobileFirst Server applications

You can use the following features for Java EE 6 or Java EE 7.

MobileFirst Server administration service
- `jdbc-4.0` (`jdbc-4.1` for Java EE 7)
- `appSecurity-2.0`
- `restConnector-1.0`
- `usr:MFPDecoderFeature-1.0`

MobileFirst Server push service
- `jdbc-4.0` (`jdbc-4.1` for Java EE 7)
- `servlet-3.0` (`servlet-3.1` for Java EE 7)
- `ssl-1.0`
- `usr:MFPDecoderFeature-1.0`

MobileFirst runtime
- `jdbc-4.0` (`jdbc-4.1` for Java EE 7)
- `servlet-3.0` (`servlet-3.1` for Java EE 7)
- `ssl-1.0`
- `usr:MFPDecoderFeature-1.0`

Global JNDI entries

The following global JNDI entries are required to configure the JMX communication between the runtime and the administration service:
- **`mfp.admin.jmx.host`**
- **`mfp.admin.jmx.port`**
- **`mfp.admin.jmx.user`**
- **`mfp.admin.jmx.pwd`**
- **`mfp.topology.platform`**
- **`mfp.topology.clustermode`**

These global JNDI entries are set with this syntax and are not prefixed by a context root. For example:

```
<jndiEntry jndiName="mfp.admin.jmx.port" value="9443"/>
```

Note: To protect against an automatic conversion of the JNDI values, so that `075` is not converted to `61` or `31.500` is not converted to `31.5`, use this syntax `'"075"'` when you define the value.
For more information about the JNDI properties for the administration service, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

For a farm configuration, see also the following topics:
- "Server farm topology" on page 6-88
- "Topologies and network flows" on page 6-78
- "Installing a server farm" on page 6-139

Class loader

For all applications, the class loader must have the parent last delegation. For example:

```
<application id="mfpadmin" name="mfpadmin" location="mfp-admin-service.war" type="war">
  [...]
  <classloader delegation="parentLast">
  </classloader>
</application>
```

Password decoder user feature

Copy the password decoder user feature to your Liberty profile. For example:

- On UNIX and Linux systems:

```
mkdir -p LIBERTY_HOME/wlp/usr/extension/lib/features
cp product_install_dir/features/com.ibm.websphere.crypto_1.0.0.jar LIBERTY_HOME/wlp/usr/extension/lib/
cp product_install_dir/features/MFPDecoderFeature-1.0.mf LIBERTY_HOME/wlp/usr/extension/lib/features/
```

- On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\extension\lib
copy /B product_install_dir\features\com.ibm.websphere.crypto_1.0.0.jar
LIBERTY_HOME\wlp\usr\extension\lib\com.ibm.websphere.crypto_1.0.0.jar
mkdir LIBERTY_HOME\wlp\usr\extension\lib\features
copy /B product_install_dir\features\MFPDecoderFeature-1.0.mf
LIBERTY_HOME\wlp\usr\extension\lib\features\MFPDecoderFeature-1.0.mf
```

MobileFirst Server administration service configuration details:

The administration service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application in the `server.xml` file.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty" on page 6-115 for the configuration details that are common to all services.

The administration service WAR file is in *`mfp_install_dir`*`/MobileFirstServer/mfp-admin-service.war`.

You can define the context root as you want. However, usually it is `/mfpadmin`.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the administration service. The following example illustrates the case to declare **`mfp.admin.push.url`** whereby the administration service is installed with `/mfpadmin` as the context root:

```
<jndiEntry jndiName="mfpadmin/mfp.admin.push.url" value="http://localhost:9080/imfpush"/>
```

If the push service is installed, you must configure the following JNDI properties:

- **`mfp.admin.push.url`**
- **`mfp.admin.authorization.server.url`**
- **`mfp.push.authorization.client.id`**
- **`mfp.push.authorization.client.secret`**
- **`mfp.admin.authorization.client.id`**
- **`mfp.admin.authorization.client.secret`**

The JNDI properties for the communication with the configuration service are as follows:

- **`mfp.config.service.user`**
- **`mfp.config.service.password`**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

Data source

The JNDI name of the data source for the administration service must be defined as `jndiName=<contextRoot>/jdbc/mfpAdminDS`. The following example illustrates the case whereby the administration service is installed with the context root `/mfpadmin`, and that the service is using a relational database:

```
<dataSource jndiName="mfpadmin/jdbc/mfpAdminDS" transactional="false">
  [...]
</dataSource>
```

Security roles

Declare the following roles in the `<application-bnd>` element of the application:

- `mfpadmin`
- `mfpdeployer`
- `mfpmonitor`
- `mfpoperator`

MobileFirst Server live update service configuration details:

The live update service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application in the `server.xml` file.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty" on page 6-115 for the configuration details that are common to all services.

The live update service WAR file is in *`mfp_install_dir`*`/MobileFirstServer/mfp-live-update.war`.

The context root of the live update service must define in this way: `/<adminContextRoot>config`. For example, if the context root of the administration service is `/mfpadmin`, then the context root of the live update service must be `/mfpadminconfig`.

Data source

The JNDI name of the data source for the live update service must be defined as `<contextRoot>/jdbc/ConfigDS`. The following example illustrates the case whereby the live update service is installed with the context root `/mfpadminconfig`, and that the service is using a relational database:

```
<dataSource jndiName="mfpadminconfig/jdbc/ConfigDS" transactional="false">
  [...]
</dataSource>
```

Security roles

Declare the `configadmin` role in the `<application-bnd>` element of the application.

At least one user must be mapped to this role. The user and its password must be provided to the following JNDI properties of the administration service:

- **`mfp.config.service.user`**
- **`mfp.config.service.password`**

MobileFirst Operations Console configuration details:

The console is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application in the `server.xml` file.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty" on page 6-115 for the configuration details that are common to all services.

The console WAR file is in *`mfp_install_dir`*`/MobileFirstServer/mfp-admin-ui.war`.

You can define the context root as you want. However, usually it is `/mfpconsole`.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the console. The following example illustrates the case to declare **`mfp.admin.endpoint`** whereby the console is installed with `/mfpconsole` as the context root:

```
<jndiEntry jndiName="mfpconsole/mfp.admin.endpoint" value="*://*:*/mfpadmin"/>
```

The typical value for the **`mfp.admin.endpoint`** property is `*://*:*/`<*`adminContextRoot`*>.

For more information about the JNDI properties, see "JNDI properties for MobileFirst Operations Console" on page 6-181.

Security roles

Declare the following roles in the `<application-bnd>` element of the application:

- `mfpadmin`
- `mfpdeployer`
- `mfpmonitor`
- `mfpoperator`

Any user that is mapped to a security role of the console must also be mapped to the same security role of the administration service.

MobileFirst runtime configuration details:

The runtime is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application in the `server.xml` file.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty" on page 6-115 for the configuration details that are common to all services.

The runtime WAR file is in *`mfp_install_dir`*`/MobileFirstServer/mfp-server.war`.

You can define the context root as you want. However, it is `/mfp` by default.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the runtime. The following example illustrates the case to declare **`mfp.analytics.url`** whereby the runtime is installed with `/mobilefirst` as the context root:

```
<jndiEntry jndiName="mobilefirst/mfp.analytics.url" value="http://localhost:9080/analytics-service/rest"/>
```

You must define the **`mobilefirst/mfp.authorization.server`** property. For example:

```
<jndiEntry jndiName="mobilefirst/mfp.authorization.server" value="embedded"/>
```

If MobileFirst Analytics is installed, you need to define the following JNDI properties:
- **`mfp.analytics.url`**
- **`mfp.analytics.console.url`**
- **`mfp.analytics.username`**
- **`mfp.analytics.password`**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst runtime" on page 6-183.

Data source

The JNDI name of the data source for the runtime must be defined as `jndiName=<contextRoot>/jdbc/mfpDS`. The following example illustrates the case whereby the runtime is installed with the context root `/mobilefirst`, and that the runtime is using a relational database:

```
<dataSource jndiName="mobilefirst/jdbc/mfpDS" transactional="false">
  [...]
</dataSource>
```

MobileFirst Server push service configuration details:

The push service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application in the `server.xml` file.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty" on page 6-115 for the configuration details that are common to all services.

The push service WAR file is in *`mfp_install_dir`*`/PushService/mfp-push-service.war`.

You must define the context root as `/imfpush`. Otherwise, the client devices cannot connect to it as the context root is hardcoded in the SDK.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the push service. The following example illustrates the case to declare **`mfp.push.analytics.user`** whereby the push service is installed with `/imfpush` as the context root:

```
<jndiEntry jndiName="imfpush/mfp.push.analytics.user" value="admin"/>
```

You need to define the following properties:

- **`mfp.push.authorization.server.url`**
- **`mfp.push.authorization.client.id`**
- **`mfp.push.authorization.client.secret`**
- **`mfp.push.services.ext.security`** - the value must be `com.ibm.mfp.push.server.security.plugin.OAuthSecurityPlugin`.
- **`mfp.push.db.type`** - for a relational database, the value must be `DB`.

If MobileFirst Analytics is configured, define the following JNDI properties:

- **`mfp.push.analytics.endpoint`**
- **`mfp.analytics.username`**
- **`mfp.analytics.password`**
- **`mfp.push.services.ext.analytics`** - the value must be `com.ibm.mfp.push.server.analytics.plugin.AnalyticsPlugin`.

For more information about the JNDI properties, see “List of JNDI properties for MobileFirst Server push service” on page 6-186.

Data source

The JNDI name of the data source for the push service must be defined as `jndiName=imfpush/jdbc/imfPushDS`.

MobileFirst Server artifacts configuration details:

The artifacts component is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application in the `server.xml` file.

Before you proceed, review “Manual installation on WebSphere Application Server Liberty” on page 6-115 for the configuration details that are common to all services.

The WAR file for this component is in *`mfp_install_dir`*`/MobileFirstServer/mfp-dev-artifacts.war`.

You must define the context root as `/mfp-dev-artifacts`.

Manual installation on WebSphere Application Server Liberty collective:

Find out more details on how to install the MobileFirst Server components on Liberty collective.

Make sure that you have also fulfilled the requirements as documented in “WebSphere Application Server Liberty prerequisites” on page 6-103.

Topology constraints

The MobileFirst Server administration service, the MobileFirst Server live update service, and MobileFirst Operations Console must be installed in a Liberty collective controller. The MobileFirst runtime and the MobileFirst Server push service must be installed in every member of the Liberty collective cluster.

The context root of the live update service must be defined as <*adminContextRoot*>`config`. The context root of the push service must be `imfpush`. For more information about the constraints, see "Constraints on the MobileFirst Server components and MobileFirst Analytics" on page 6-84.

Application server settings

You must configure the `<webContainer>` element to load the servlets immediately. This setting is required for the initialization through JMX. For example:

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

Optionally, to avoid timeout issues that break the startup sequence of the runtime and the administration service on some Liberty versions, change the default `<executor>` element. Set large values to the **`coreThreads`** and **`maxThreads`** attributes. For example:

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

You might also configure the `<tcpOptions>` element and set the **`soReuseAddr`** attribute to `true`.

```
<tcpOptions soReuseAddr="true"/>
```

Liberty features required by the MobileFirst Server applications

You need to add the following features for Java EE 6 or Java EE 7.

MobileFirst Server administration service
- `jdbc-4.0` (`jdbc-4.1` for Java EE 7)
- `appSecurity-2.0`
- `restConnector-1.0`
- `usr:MFPDecoderFeature-1.0`

MobileFirst Server push service
- `jdbc-4.0` (`jdbc-4.1` for Java EE 7)
- `servlet-3.0` (`servlet-3.1` for Java EE 7)
- `ssl-1.0`
- `usr:MFPDecoderFeature-1.0`

MobileFirst runtime
- `jdbc-4.0` (`jdbc-4.1` for Java EE 7)
- `servlet-3.0` (`servlet-3.1` for Java EE 7)
- `ssl-1.0`
- `usr:MFPDecoderFeature-1.0`

JNDI entries

The following global JNDI entries are required to configure the JMX communication between the runtime and the administration service:

- **mfp.admin.jmx.host**
- **mfp.admin.jmx.port**
- **mfp.admin.jmx.user**
- **mfp.admin.jmx.pwd**
- **mfp.topology.platform**
- **mfp.topology.clustermode**
- **mfp.admin.serverid**

These global JNDI entries are set with this syntax and are not prefixed by a context root. For example:

```
<jndiEntry jndiName="mfp.admin.jmx.port" value="9443"/>
```

Note: To protect against an automatic conversion of the JNDI values, so that `075` is not converted to `61` or `31.500` is not converted to `31.5`, use this syntax `'"075"'` when you define the value.
For more information about the JNDI properties for the administration service, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

For more information about the JNDI properties for the runtime, see "List of JNDI properties for MobileFirst runtime" on page 6-183.

Class loader

For all applications, the class loader must have the parent last delegation. For example:

```
<application id="mfpadmin" name="mfpadmin" location="mfp-admin-service.war" type="war">
  [...]
  <classloader delegation="parentLast">
  </classloader>
</application>
```

Password decoder user feature

Copy the password decoder user feature to your Liberty profile. For example:

- On UNIX and Linux systems:

```
mkdir -p LIBERTY_HOME/wlp/usr/extension/lib/features
cp product_install_dir/features/com.ibm.websphere.crypto_1.0.0.jar LIBERTY_HOME/wlp/usr/extension/lib/
cp product_install_dir/features/MFPDecoderFeature-1.0.mf LIBERTY_HOME/wlp/usr/extension/lib/features/
```

- On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\extension\lib
copy /B product_install_dir\features\com.ibm.websphere.crypto_1.0.0.jar
LIBERTY_HOME\wlp\usr\extension\lib\com.ibm.websphere.crypto_1.0.0.jar
mkdir LIBERTY_HOME\wlp\usr\extension\lib\features
copy /B product_install_dir\features\MFPDecoderFeature-1.0.mf
LIBERTY_HOME\wlp\usr\extension\lib\features\MFPDecoderFeature-1.0.mf
```

MobileFirst Server administration service configuration details:

The administration service is packaged as a WAR application for you to deploy to the Liberty collective controller. You need to make some specific configurations for this application in the `server.xml` file of the Liberty collective controller.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty collective" on page 6-121 for the configuration details that are common to all services.

The administration service WAR file is in *mfp_install_dir*/MobileFirstServer/mfp-admin-service.war.

You can define the context root as you want. However, it is /mfpadmin by default.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the administration service. The following example illustrates the case to declare **mfp.admin.push.url** whereby the administration service is installed with /mfpadmin as the context root:

```
<jndiEntry jndiName="mfpadmin/mfp.admin.push.url" value="http://localhost:9080/imfpush"/>
```

If the push service is installed, you must configure the following JNDI properties:
- **mfp.admin.push.url**
- **mfp.admin.authorization.server.url**
- **mfp.push.authorization.client.id**
- **mfp.push.authorization.client.secret**
- **mfp.admin.authorization.client.id**
- **mfp.admin.authorization.client.secret**

The JNDI properties for the communication with the configuration service are as follows:
- **mfp.config.service.user**
- **mfp.config.service.password**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

Data source

The JNDI name of the data source for the administration service must be defined as jndiName=<*contextRoot*>/jdbc/mfpAdminDS. The following example illustrates the case whereby the administration service is installed with the context root /mfpadmin, and that the service is using a relational database:

```
<dataSource jndiName="mfpadmin/jdbc/mfpAdminDS" transactional="false">
  [...]
</dataSource>
```

Security roles

Declare the following roles in the <application-bnd> element of the application:
- mfpadmin
- mfpdeployer
- mfpmonitor
- mfpoperator

MobileFirst Server live update service configuration details:

The live update service is packaged as a WAR application for you to deploy to the Liberty collective controller. You need to make some specific configurations for this application in the `server.xml` file of the Liberty collective controller.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty collective" on page 6-121 for the configuration details that are common to all services.

The live update service WAR file is in *mfp_install_dir*`/MobileFirstServer/mfp-live-update.war`.

The context root of the live update service must define in this way: `/<`*adminContextRoot*`>config`. For example, if the context root of the administration service is `/mfpadmin`, then the context root of the live update service must be `/mfpadminconfig`.

Data source

The JNDI name of the data source for the live update service must be defined as `<`*contextRoot*`>/jdbc/ConfigDS`. The following example illustrates the case whereby the live update service is installed with the context root `/mfpadminconfig`, and that the service is using a relational database:

```
<dataSource jndiName="mfpadminconfig/jdbc/ConfigDS" transactional="false">
  [...]
</dataSource>
```

Security roles

Declare the `configadmin` role in the `<application-bnd>` element of the application.

At least one user must be mapped to this role. The user and its password must be provided to the following JNDI properties of the administration service:

- **`mfp.config.service.user`**
- **`mfp.config.service.password`**

MobileFirst Operations Console configuration details:

The console is packaged as a WAR application for you to deploy to the Liberty collective controller. You need to make some specific configurations for this application in the `server.xml` file of the Liberty collective controller.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty collective" on page 6-121 for the configuration details that are common to all services.

The console WAR file is in *mfp_install_dir*`/MobileFirstServer/mfp-admin-ui.war`.

You can define the context root as you want. However, it is `/mfpconsole` by default.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the console. The following example illustrates the case to declare **mfp.admin.endpoint** whereby the console is installed with /mfpconsole as the context root:

```
<jndiEntry jndiName="mfpconsole/mfp.admin.endpoint" value="*://*:*/mfpadmin"/>
```

The typical value for the **mfp.admin.endpoint** property is `*://*:*/<adminContextRoot>`.

For more information about the JNDI properties, see "JNDI properties for MobileFirst Operations Console" on page 6-181.

Security roles

Declare the following roles in the `<application-bnd>` element of the application:

- mfpadmin
- mfpdeployer
- mfpmonitor
- mfpoperator

Any user that is mapped to a security role of the console must also be mapped to the same security role of the administration service.

MobileFirst runtime configuration details:

The runtime is packaged as a WAR application for you to deploy to the Liberty collective cluster members. You need to make some specific configurations for this application in the `server.xml` file of every Liberty collective cluster member.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty collective" on page 6-121 for the configuration details that are common to all services.

The runtime WAR file is in *mfp_install_dir*/MobileFirstServer/mfp-server.war.

You can define the context root as you want. However, it is /mfp by default.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the runtime. The following example illustrates the case to declare **mfp.analytics.url** whereby the runtime is installed with /mobilefirst as the context root:

```
<jndiEntry jndiName="mobilefirst/mfp.analytics.url" value="http://localhost:9080/analytics-service/rest"/>
```

You must define the **mobilefirst/mfp.authorization.server** property. For example:

```
<jndiEntry jndiName="mobilefirst/mfp.authorization.server" value="embedded"/>
```

If MobileFirst Analytics is installed, you need to define the following JNDI properties:

- **mfp.analytics.url**
- **mfp.analytics.console.url**

- **mfp.analytics.username**
- **mfp.analytics.password**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst runtime" on page 6-183.

Data source

The JNDI name of the data source for the runtime must be defined as jndiName=<*contextRoot*>/jdbc/mfpDS. The following example illustrates the case whereby the runtime is installed with the context root /mobilefirst, and that the runtime is using a relational database:

```
<dataSource jndiName="mobilefirst/jdbc/mfpDS" transactional="false">
  [...]
</dataSource>
```

MobileFirst Server push service configuration details:

The push service is packaged as a WAR application for you to deploy to a Liberty collective cluster member or to a Liberty server.

If you install the push service in a Liberty server, see "MobileFirst Server push service configuration details" on page 6-120 under "Manual installation on WebSphere Application Server Liberty" on page 6-115.

When the MobileFirst Server push service is installed in a Liberty collective, it can be installed in the same cluster than the runtime or in another cluster.

You need to make some specific configurations for this application in the server.xml file of every Liberty collective cluster member.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty collective" on page 6-121 for the configuration details that are common to all services.

The push service WAR file is in *mfp_install_dir*/PushService/mfp-push-service.war.

You must define the context root as /imfpush. Otherwise, the client devices cannot connect to it as the context root is hardcoded in the SDK.

Mandatory JNDI properties

When you define the JNDI properties, the JNDI names must be prefixed with the context root of the push service. The following example illustrates the case to declare **mfp.push.analytics.user** whereby the push service is installed with /imfpush as the context root:

```
<jndiEntry jndiName="imfpush/mfp.push.analytics.user" value="admin"/>
```

You need to define the following properties:

- **mfp.push.authorization.server.url**
- **mfp.push.authorization.client.id**
- **mfp.push.authorization.client.secret**
- **mfp.push.services.ext.security** - the value must be com.ibm.mfp.push.server.security.plugin.OAuthSecurityPlugin.

- **mfp.push.db.type** - for a relational database, the value must be DB.

If MobileFirst Analytics is configured, define the following JNDI properties:

- **mfp.push.analytics.endpoint**
- **mfp.analytics.username**
- **mfp.analytics.password**
- **mfp.push.services.ext.analytics** - the value must be com.ibm.mfp.push.server.analytics.plugin.AnalyticsPlugin.

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server push service" on page 6-186.

Data source

The JNDI name of the data source for the push service must be defined as jndiName=imfpush/jdbc/imfPushDS.

MobileFirst Server artifacts configuration details:

The artifacts component is packaged as a WAR application for you to deploy to the Liberty collective controller. You need to make some specific configurations for this application in the server.xml file of the Liberty collective controller.

Before you proceed, review "Manual installation on WebSphere Application Server Liberty collective" on page 6-121 for the configuration details that are common to all services.

The WAR file for this component is in *mfp_install_dir*/MobileFirstServer/mfp-dev-artifacts.war.

You must define the context root as /mfp-dev-artifacts.

Manual installation on Apache Tomcat:

Find out more details on how to install the MobileFirst Server components on Apache Tomcat.

Make sure that you have fulfilled the requirements as documented in "Apache Tomcat prerequisites" on page 6-100.

Application server settings

You must activate the Single Sign On Valve. For example:

```
<Valve className="org.apache.catalina.authenticator.SingleSignOn"/>
```

Optionally, you might want to activate the memory realm if the users are defined in tomcat-users.xml. For example:

```
<Realm className="org.apache.catalina.realm.MemoryRealm"/>
```

Topology constraints

The MobileFirst Server administration service, the MobileFirst Server live update service, and the MobileFirst runtime must be installed on the same application server. The context root of the live update service must be defined as <*adminContextRoot*>config. The context root of the push service must be imfpush.

For more information about the constraints, see "Constraints on the MobileFirst Server components and MobileFirst Analytics" on page 6-84.

MobileFirst Server administration service configuration details:

The administration service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on Apache Tomcat" on page 6-128 for the configuration details that are common to all services.

The administration service WAR file is in *mfp_install_dir*/MobileFirstServer/mfp-admin-service.war.

You can define the context root as you want. However, usually it is /mfpadmin.

Mandatory JNDI properties

The JNDI properties are defined within the <Environment> element in the application context. For example:

```
<Environment name="mfp.admin.push.url" value="http://localhost:8080/imfpush" type="java.lang.Strir
```

To enable the JMX communication with the runtime, define the following JNDI properties:

- **mfp.topology.platform**
- **mfp.topology.clustermode**

If the push service is installed, you must also configure the following JNDI properties:

- **mfp.admin.push.url**
- **mfp.admin.authorization.server.url**
- **mfp.push.authorization.client.id**
- **mfp.push.authorization.client.secret**
- **mfp.admin.authorization.client.id**
- **mfp.admin.authorization.client.secret**

The JNDI properties for the communication with the live update service are as follows:

- **mfp.config.service.user**
- **mfp.config.service.password**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

Data source

The data source (jdbc/mfpAdminDS) is declared as a resource in the <Context> element. For example:

```
<Resource name="jdbc/mfpAdminDS" type="javax.sql.DataSource" .../>
```

Security roles

The security roles available for the administration service application are:
- `mfpadmin`
- `mfpdeployer`
- `mfpmonitor`
- `mfpoperator`

MobileFirst Server live update service configuration details:

The live update service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review “Manual installation on Apache Tomcat” on page 6-128 for the configuration details that are common to all services.

The live update service WAR file is in *`mfp_install_dir`*`/MobileFirstServer/mfp-live-update.war`.

The context root of the live update service must define in this way: `/`*`<adminContextRoot>`*`config`. For example, if the context root of the administration service is `/mfpadmin`, then the context root of the live update service must be `/mfpadminconfig`.

Data source

The JNDI name of the data source for the live update service must be defined as `jdbc/ConfigDS`. Declare it as a resource in the `<Context>` element.

Security roles

The security role available for the live update service application is `configadmin`.

At least one user must be mapped to this role. The user and its password must be provided to the following JNDI properties of the administration service:
- **`mfp.config.service.user`**
- **`mfp.config.service.password`**

MobileFirst Operations Console configuration details:

The console is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review “Manual installation on Apache Tomcat” on page 6-128 for the configuration details that are common to all services.

The console WAR file is in *`mfp_install_dir`*`/MobileFirstServer/mfp-admin-ui.war`.

You can define the context root as you want. However, usually it is `/mfpconsole`.

Mandatory JNDI properties

You need to define the **`mfp.admin.endpoint`** property. The typical value for this property is `*://*:*/`*`<adminContextRoot>`*.

For more information about the JNDI properties, see "JNDI properties for MobileFirst Operations Console" on page 6-181.

Security roles

The security roles available for the application are:
- mfpadmin
- mfpdeployer
- mfpmonitor
- mfpoperator

MobileFirst runtime configuration details:

The runtime is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on Apache Tomcat" on page 6-128 for the configuration details that are common to all services.

The runtime WAR file is in *mfp_install_dir*/MobileFirstServer/mfp-server.war.

You can define the context root as you want. However, it is /mfp by default.

Mandatory JNDI properties

You must define the **mfp.authorization.server** property. For example:

```
<Environment name="mfp.authorization.server" value="embedded" type="java.lang.String" override="fa
```

To enable the JMX communication with the administration service, define the following JNDI properties:
- **mfp.topology.platform**
- **mfp.topology.clustermode**

If MobileFirst Analytics is installed, you also need to define the following JNDI properties:
- **mfp.analytics.url**
- **mfp.analytics.console.url**
- **mfp.analytics.username**
- **mfp.analytics.password**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst runtime" on page 6-183.

Data source

The JNDI name of the data source for the runtime must be defined as jdbc/mfpDS. Declare it as a resource in the <Context> element.

MobileFirst Server push service configuration details:

The push service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on Apache Tomcat" on page 6-128 for the configuration details that are common to all services.

The push service WAR file is in *mfp_install_dir*/PushService/mfp-push-service.war.

You must define the context root as /imfpush. Otherwise, the client devices cannot connect to it as the context root is hardcoded in the SDK.

Mandatory JNDI properties

You need to define the following properties:

- **mfp.push.authorization.server.url**
- **mfp.push.authorization.client.id**
- **mfp.push.authorization.client.secret**
- **mfp.push.services.ext.security** - the value must be com.ibm.mfp.push.server.security.plugin.OAuthSecurityPlugin.
- **mfp.push.db.type** - for a relational database, the value must be DB.

If MobileFirst Analytics is configured, define the following JNDI properties:

- **mfp.push.analytics.endpoint**
- **mfp.analytics.username**
- **mfp.analytics.password**
- **mfp.push.services.ext.analytics** - the value must be com.ibm.mfp.push.server.analytics.plugin.AnalyticsPlugin.

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server push service" on page 6-186.

Data source

The JNDI name of the data source for the push service must be defined as jdbc/imfPushDS.

MobileFirst Server artifacts configuration details:

The artifacts component is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on Apache Tomcat" on page 6-128 for the configuration details that are common to all services.

The WAR file for this component is in *mfp_install_dir*/MobileFirstServer/mfp-dev-artifacts.war.

You must define the context root as /mfp-dev-artifacts.

Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment:

Find out more details on how to install the MobileFirst Server components on WebSphere Application Server and WebSphere Application Server Network Deployment.

Make sure that you have fulfilled the requirements as documented in "WebSphere Application Server and WebSphere Application Server Network Deployment prerequisites" on page 6-104.

Topology constraints

On a stand-alone WebSphere Application Server
The MobileFirst Server administration service, the MobileFirst Server live update service, and the MobileFirst runtime must be installed on the same application server. The context root of the live update service must be defined as `<`*`adminContextRoot`*`>config`. The context root of the push service must be `imfpush`. For more information about the constraints, see "Constraints on the MobileFirst Server components and MobileFirst Analytics" on page 6-84.

On WebSphere Application Server Network Deployment
The deployment manager must be running while MobileFirst Server is running. The deployment manager is used for the JMX communication between the runtime and the administration service. The administration service and the live update service must be installed on the same application server. The runtime can be installed on different servers than the administration service, but it must be on the same cell.

Application server settings

The administrative security and the application security must be enabled. You can enable the application security in the WebSphere Application Server administration console:

1. Log in to the WebSphere Application Server administration console.
2. Click **Security** > **Global Security**. Ensure that `Enable administrative security` is selected.
3. Also, ensure that `Enable application security` is selected. The application security can be enabled only if administrative security is enabled.
4. Click **OK**.
5. Save the changes.

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

The server class loader policy must support parent last delegation. The MobileFirst Server WAR files must be installed with parent last class loader mode. Review the class-loader policy:

1. Log in to the WebSphere Application Server administration console.
2. Click **Servers** > **Server Types** > **WebSphere application servers**, and click on the server that is used for IBM MobileFirst Platform Foundation.
3. If the class-loader policy is set to `Multiple`, do nothing.
4. If the class-loader policy is set to `Single` and the class loading mode is set to `Classes loaded with local class loader first (parent last)`, do nothing.
5. If the class-loader policy is set to `Single` and the class loading mode is set to `Classes loaded with parent class loader first` (parent first), change the class-loader policy to `Multiple`. Also, set the class loader order of all applications other than MobileFirst Server applications to `Classes loaded with parent class loader first` (parent first).

Class loader

For all MobileFirst Server applications, the class loader must have the parent last delegation.

To set the class loader delegation to parent last after an application is installed, follow these steps:

1. Click the **Manage Applications** link, or click **Applications** > **Application Types** > **WebSphere entreprise applications**.
2. Click the MobileFirst Server application. By default the name of the application is the name of the WAR file.
3. In the **Detail Properties** section, click the **Class loading and update detection** link.
4. In the **Class loader order** pane, select the `Classes loaded with local class loader first (parent last)` option.
5. Click **OK**.
6. In the **Modules** section, click the **Manage Modules** link.
7. Click the module.
8. For the **Class loader order** field, select the `Classes loaded with local class loader first (parent last)` option.
9. Click **OK** twice to confirm the selection and back to the **Configuration** panel of the application.
10. Click **Save** to persist the changes.

MobileFirst Server administration service configuration details:

The administration service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment" on page 6-132 for the configuration details that are common to all services.

The administration service WAR file is in *mfp_install_dir*`/MobileFirstServer/mfp-admin-service.war`.

You can define the context root as you want. However, usually it is `/mfpadmin`.

Mandatory JNDI properties

You can set JNDI properties with the WebSphere Application Server administration console. Go to **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for Web modules** and set the entries.

To enable the JMX communication with the runtime, you must configure the following JNDI properties:

On WebSphere Application Server Network Deployment

- **`mfp.admin.jmx.dmgr.host`**
- **`mfp.admin.jmx.dmgr.port`** - the SOAP port on the deployment manager.
- **`mfp.topology.platform`** - set the value as `WAS`.

- **mfp.topology.clustermode** - set the value as `Cluster`.
- **mfp.admin.jmx.connector** - set the value as `SOAP`.

On a stand-alone WebSphere Application Server

- **mfp.topology.platform** - set the value as `WAS`.
- **mfp.topology.clustermode** - set the value as `Standalone`.
- **mfp.admin.jmx.connector** - set the value as `SOAP`.

If the push service is installed, you must configure the following JNDI properties:

- **mfp.admin.push.url**
- **mfp.admin.authorization.server.url**
- **mfp.push.authorization.client.id**
- **mfp.push.authorization.client.secret**
- **mfp.admin.authorization.client.id**
- **mfp.admin.authorization.client.secret**

The JNDI properties for the communication with the configuration service are as follows:

- **mfp.config.service.user**
- **mfp.config.service.password**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

Data source

Create a data source for the administration service and map it to `jdbc/mfpAdminDS`.

Start order

The administration service application must start before the runtime application. You can set the order at **Startup behavior** section. For example, set the **Startup Order** to `1` for the administration service and `2` to the runtime.

Security roles

The following roles are defined for this application:

- `mfpadmin`
- `mfpdeployer`
- `mfpmonitor`
- `mfpoperator`

MobileFirst Server live update service configuration details:

The live update service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment" on page 6-132 for the configuration details that are common to all services.

The live update service WAR file is in *mfp_install_dir*/MobileFirstServer/mfp-live-update.war.

The context root of the live update service must define in this way: /*<adminContextRoot>*config. For example, if the context root of the administration service is /mfpadmin, then the context root of the live update service must be /mfpadminconfig.

Data source

Create a data source for the live update service and map it to jdbc/ConfigDS.

Security roles

The configadmin role is defined for this application.

At least one user must be mapped to this role. The user and its password must be provided to the following JNDI properties of the administration service:
- **mfp.config.service.user**
- **mfp.config.service.password**

MobileFirst Operations Console configuration details:

The console is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment" on page 6-132 for the configuration details that are common to all services.

The console WAR file is in *mfp_install_dir*/MobileFirstServer/mfp-admin-ui.war.

You can define the context root as you want. However, usually it is /mfpconsole.

Mandatory JNDI properties

You can set JNDI properties with the WebSphere Application Server administration console. Go to **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for Web modules** and set the entries.

You need to define the **mfp.admin.endpoint** property. The typical value for this property is *://*:*/*<adminContextRoot>*.

For more information about the JNDI properties, see "JNDI properties for MobileFirst Operations Console" on page 6-181.

Security roles

The following roles are defined for the application:
- mfpadmin
- mfpdeployer
- mfpmonitor
- mfpoperator

Any user that is mapped to a security role of the console must also be mapped to the same security role of the administration service.

MobileFirst runtime configuration details:

The runtime is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment" on page 6-132 for the configuration details that are common to all services.

The runtime WAR file is in `mfp_install_dir/MobileFirstServer/mfp-server.war`.

You can define the context root as you want. However, it is `/mfp` by default.

Mandatory JNDI properties

You can set JNDI properties with the WebSphere Application Server administration console. Go to **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for Web modules** and set the entries.

You must define the **`mfp.authorization.server`** property with the value as `embedded`.

Also, define the following JNDI properties to enable the JMX communication with the administration service:

On WebSphere Application Server Network Deployment
- **`mfp.admin.jmx.dmgr.host`** - the host name of the deployment manager.
- **`mfp.admin.jmx.dmgr.port`** - the SOAP port of the deployment manager.
- **`mfp.topology.platform`** - set the value as `WAS`.
- **`mfp.topology.clustermode`** - set the value as `Cluster`.
- **`mfp.admin.jmx.connector`** - set the value as `SOAP`.

On a stand-alone WebSphere Application Server
- **`mfp.topology.platform`** - set the value as `WAS`.
- **`mfp.topology.clustermode`** - set the value as `Standalone`.
- **`mfp.admin.jmx.connector`** - set the value as `SOAP`.

If MobileFirst Analytics is installed, you also need to define the following JNDI properties:
- **`mfp.analytics.url`**
- **`mfp.analytics.console.url`**
- **`mfp.analytics.username`**
- **`mfp.analytics.password`**

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst runtime" on page 6-183.

Start order

The runtime application must start after the administration service application. You can set the order at **Startup behavior** section. For example, set the **Startup Order** to 1 for the administration service and 2 to the runtime.

Data source

Create a data source for the runtime and map it to `jdbc/mfpDS`.

MobileFirst Server push service configuration details:

The push service is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review “Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment” on page 6-132 for the configuration details that are common to all services.

The push service WAR file is in *mfp_install_dir*/PushService/mfp-push-service.war.

You must define the context root as `/imfpush`. Otherwise, the client devices cannot connect to it as the context root is hardcoded in the SDK.

Mandatory JNDI properties

You can set JNDI properties with the WebSphere Application Server administration console. Go to **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for Web modules** and set the entries.

You need to define the following properties:
- **mfp.push.authorization.server.url**
- **mfp.push.authorization.client.id**
- **mfp.push.authorization.client.secret**
- **mfp.push.services.ext.security** - the value must be `com.ibm.mfp.push.server.security.plugin.OAuthSecurityPlugin`.
- **mfp.push.db.type** - for a relational database, the value must be `DB`.

If MobileFirst Analytics is configured, define the following JNDI properties:
- **mfp.push.analytics.endpoint**
- **mfp.analytics.username**
- **mfp.analytics.password**
- **mfp.push.services.ext.analytics** - the value must be `com.ibm.mfp.push.server.analytics.plugin.AnalyticsPlugin`.

For more information about the JNDI properties, see “List of JNDI properties for MobileFirst Server push service” on page 6-186.

Data source

Create the data source for the push service and map it to `jdbc/imfPushDS`.

MobileFirst Server artifacts configuration details:

The artifacts component is packaged as a WAR application for you to deploy to the application server. You need to make some specific configurations for this application.

Before you proceed, review "Manual installation on WebSphere Application Server and WebSphere Application Server Network Deployment" on page 6-132 for the configuration details that are common to all services.

The WAR file for this component is in `mfp_install_dir/MobileFirstServer/mfp-dev-artifacts.war`.

You must define the context root as `/mfp-dev-artifacts`.

Installing a server farm:

You can install your server farm by running Ant tasks, with the Server Configuration Tool, or manually.

Planning the configuration of a server farm:

To plan the configuration of a server farm, choose the application server, configure the MobileFirst databases, and deploy the WAR files of the MobileFirst Server components on each server of the farm. You have the options to use the Server Configuration Tool, Ant tasks, or manual operations to configure a server farm.

When you intend to plan a server farm installation, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84 first, and in particular see "Server farm topology" on page 6-88.

In IBM MobileFirst Platform Foundation, a server farm is composed of multiple stand-alone application servers that are not federated or administered by a managing component of an application server. 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 Deployment.
- 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 MobileFirst Operations Console or through the MobileFirst Server administration service 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 an app or of an adapter. The replication is done via JMX calls performed by the administration service 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" on page 6-139, 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.

Installing a server farm with the Server Configuration Tool:

Use the Server Configuration Tool to configure each server in the farm according to the requirements of the single type of application server that is used for each member of the server farm.

About this task

When you plan a server farm with the Server Configuration Tool, first create the stand-alone servers and configure their respective truststores so that they can communicate with one another in a secure way. Then, run the tool that does the following operations:

- Configure the database instance that is shared by the MobileFirst Server components.
- Deploy the MobileFirst Server components to each server
- Modify its configuration to make it a member of a server farm

Procedure

1. Prepare the application servers that must be configured as the server farm members.
   a. Choose the type of application server to use to configure the members of the server farm. IBM MobileFirst Platform Foundation supports the following 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, only the SOAP connector is supported by IBM MobileFirst Platform Foundation.
      - WebSphere Application Server Liberty profile
      - Apache Tomcat

      To know which versions of the application servers are supported, see "System requirements" on page 2-7.

      Important:

      IBM MobileFirst Platform Foundation supports only homogeneous server farms. A server farm is homogeneous when it connects same type of application servers. 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.

   b. 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 that is configured on the same host. This constraint applies to the ports used by HTTP, HTTPS, REST, SOAP, and RMI protocols.

      Each of these servers must have the MobileFirst Server administration service, the MobileFirst Server live update service, and one or more MobileFirst runtimes deployed on it.

      For more information about setting up a server, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84.
   c. Exchange the signer certificates between all the servers in their respective truststores.

      This step is mandatory for the farms that use WebSphere Application Server full profile or Liberty as security must be enabled. In addition, for Liberty farms, the same LTPA configuration must be replicated on each server to ensure single-sign on capability. To do this configuration, follow the guidelines in step 6 on page 6-147 of "Configuring a server farm manually" on page 6-144.
2. Run the Server Configuration Tool for each server of the farm.

   All servers must share the same databases. Make sure to select the deployment type: `Server farm deployment` in the **Application Server Settings** panel. For more information about the tool, see "Running the Server Configuration Tool" on page 6-106.

Installing a server farm with Ant tasks:

Use Ant tasks to configure each server in the farm according to the requirements of the single type of application server that is used for each member of the server farm.

About this task

When you plan a server farm with Ant tasks, first create the stand-alone servers and configure their respective truststores so that they can communicate with one another in a secure way. Then, run Ant tasks to configure the database instance that is shared by the MobileFirst Server components. Finally, run Ant tasks to deploy the MobileFirst Server components to each server and modify its configuration to make it a member of a server farm.

Procedure
1. Prepare the application servers that must be configured as the server farm members.
   a. Choose the type of application server to use to configure the members of the server farm. IBM MobileFirst Platform Foundation supports the following 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, only the SOAP connector is supported by IBM MobileFirst Platform Foundation.

- WebSphere Application Server Liberty profile
- Apache Tomcat

To know which versions of the application servers are supported, see "System requirements" on page 2-7.

Important:

IBM MobileFirst Platform Foundation supports only homogeneous server farms. A server farm is homogeneous when it connects same type of application servers. 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.

b. 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 that is configured on the same host. This constraint applies to the ports used by HTTP, HTTPS, REST, SOAP, and RMI protocols.

   Each of these servers must have the MobileFirst Server administration service, the MobileFirst Server live update service, and one or more MobileFirst runtimes deployed on it.

   For more information about setting up a server, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84.

c. Exchange the signer certificates between all the servers in their respective truststores.

   This step is mandatory for the farms that use WebSphere Application Server full profile or Liberty as security must be enabled. In addition, for Liberty farms, the same LTPA configuration must be replicated on each server to ensure single-sign on capability. To do this configuration, follow the guidelines in step 6 on page 6-147 of "Configuring a server farm manually" on page 6-144.

2. Configure the database for the administration service, the live update service, and the runtime.

   a. Decide which database that you want to use and choose the Ant file to create and configure the database in the *`mfp_install_dir`*`/MobileFirstServer/configuration-samples` directory:

      - For DB2, use `create-database-db2.xml`.
      - For MySQL, use `create-database-mysql.xml`.
      - For Oracle, use `create-database-oracle.xml`.

      Note: Do not use the Derby database in a farm topology because the Derby database allows only a single connection at a time.

   b. Edit the Ant file and enter all the required properties for the database.

      To enable the configuration of the database that is used by the MobileFirst Server components, set the values of the following properties:

      - Set **`mfp.process.admin`** to `true`. To configure the database for the administration service and the live update service.

- Set **`mfp.process.runtime`** to `true`. To configure the database for the runtime.

c. Run the following commands from the *`mfp_install_dir`*`/MobileFirstServer/configuration-samples` directory where *`create-database-ant-file`*`.xml` must be replaced with the actual Ant file name that you chose:

```
mfp_install_dir/shortcuts/ant -f create-database-ant-file.xml
admdatabases
```

```
mfp_install_dir/shortcuts/ant -f create-database-ant-file.xml
rtmdatabases
```

As the MobileFirst Server databases are shared between the application servers in a farm, these two commands must be run only once, whatever the number of servers in the farm.

d. Optionally, if you want to install another runtime, you must configure another database with another database name or schema.

To do so, edit the Ant file, modify the properties, and run the following command once, whatever the number of servers in the farm:

```
mfp_install_dir/shortcuts/ant -f create-database-ant-file.xml
rtmdatabases
```

3. Deploy the administration service, the live update service, and the runtime on the servers and configure these servers as the members of a server farm.

a. Choose the Ant file that corresponds to your application server and your database in the *`mfp_install_dir`*`/MobileFirstServer/configuration-samples` directory to deploy the administration service, the live update service, and the runtime on the servers.

For example, choose the `configure-liberty-db2.xml` file for a deployment on Liberty server with the DB2 database. Make as many copies of this file as the number of members that you want in the farm.

Note: Keep these files after the configuration as they can be reused for upgrading the MobileFirst Server components that are already deployed, or for uninstalling them from each member of the farm.

b. Edit each copy of the Ant file, enter the same properties for the database that are used at step 2 on page 6-142, and also enter the other required properties for the application server.

To configure the server as a server farm member, set the values of the following properties:

- Set **`mfp.farm.configure`** to `true`.
- **`mfp.farm.server.id`**: An identifier that you define for this farm member. Make sure that each server in the farm has its own unique identifier. If two servers in the farm have the same identifier, the farm might behave in an unpredictable way.
- **`mfp.config.service.user`**: The user name that is used to access the live update service. The user name must be the same for all the members of the farm.
- **`mfp.config.service.password`**: The password that is used to access the live update service. The password must be the same for all the members of the farm.

To enable the deployment of the WAR files of the MobileFirst Server components on the server, set the values of the following properties:

- Set **mfp.process.admin** to true. To deploy the WAR files of the administration service and the live update service.
- Set **mfp.process.runtime** to true. To deploy the WAR file of the runtime.

Note: If you plan to install more than one runtime on the servers of the farm, specify the attribute **id** and set a value that must be unique for each runtime on the **installmobilefirstruntime**, **updatemobilefirstruntime**, and **uninstallmobilefirstruntime** Ant tasks.
For example,

```
<target name="rtminstall">
    <installmobilefirstruntime execute="true" contextroot="/runtime1" id="rtm1">
```

c. For each server, run the following commands where *configure-appserver-database-ant-file*.xml must be replaced with the actual Ant file name that you chose:

mfp_install_dir/shortcuts/ant -f *configure-appserver-database-ant-file*.xml adminstall

mfp_install_dir/shortcuts/ant -f *configure-appserver-database-ant-file*.xml rtminstall

These commands run the **installmobilefirstadmin** and **installmobilefirstruntime** Ant tasks. For more information about these tasks, see "Ant tasks for installation of MobileFirst Operations Console, MobileFirst Server artifacts, MobileFirst Server administration, and live update services" on page 6-274 and "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

d. Optionally, if you want to install another runtime, do the following steps:

1) Make a copy of the Ant file that you configured at step 3b on page 6-143.
2) Edit the copy, set a distinct context root, and a value for the attribute **id** of **installmobilefirstruntime**, **updatemobilefirstruntime**, and **uninstallmobilefirstruntime** that is different from the other runtime configuration.
3) Run the following command on each server on the farm where *configure-appserver-database-ant-file2*.xml must be replaced with the actual name of the Ant file that is edited:

   mfp_install_dir/shortcuts/ant -f *configure-appserver-database-ant-file2*.xml rtminstall
4) Repeat this step for each server of the farm.

4. Restart all the servers.

Configuring a server farm manually:

You must configure each server in the farm according to the requirements of the single type of application server that is 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 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 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 2-7.

   Note: IBM MobileFirst Platform Foundation 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 Server 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 "Setting up databases" on page 6-63.
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 that is configured on the same host. This constraint applies to the ports used by HTTP, HTTPS, REST, SOAP, and RMI protocols.

   Each of these servers must have the MobileFirst Server administration service, the MobileFirst Server live update service, and one or more MobileFirst runtimes deployed on it.

   For more information about setting up a server, see "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84.

   When each of these servers is working properly in a stand-alone topology, you can transform them into members 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. In a server farm topology, the `mfp.config.service.user` and `mfp.config.service.password` JNDI properties must have the same value for all the members of the farm. 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="mfp.topology.clustermode" value="Farm"/>
<jndiEntry jndiName="mfp.admin.serverid" value="farm_member_1"/>
<jndiEntry jndiName="mfp.admin.jmx.user" value="myRESTConnectorUser"/>
<jndiEntry jndiName="mfp.admin.jmx.pwd" value="password-of-rest-connector-user"/>
<jndiEntry jndiName="mfp.admin.jmx.host" value="93.12.0.12"/>
<jndiEntry jndiName="mfp.admin.jmx.port" value="9443"/>
```

These properties must be set with appropriate values:

- **mfp.admin.serverid**: The identifier that you defined for this farm member. This identifier must be unique across all farm members.
- **mfp.admin.jmx.user** and **mfp.admin.jmx.pwd**: These values must match the credentials of a user as declared in the `<administrator-role/>` element.
- **mfp.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.
- **mfp.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 administration service context and in every runtime context.

```
<Environment name="mfp.topology.clustermode" value="Farm" type="java.lang.String" override="false"/>
<Environment name="mfp.admin.serverid" value="farm_member_1" type="java.lang.String" override="false"/>
```

The **mfp.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`. In addition, you must make sure that the `-Dcom.sun.management.jmxremote.port` JVM argument is set with a port that is not already in use to enable JMX RMI connections. Both arguments are set in the *CATALINA_OPTS* environment variable.

• **WebSphere Application Server full profile**

You must declare the following JNDI properties in the administration service and in every runtime application deployed on the server.

- `mfp.topology.clustermode`
- `mfp.admin.serverid`

a. In the WebSphere Application Server console, select **Applications** > **Application Types** > **WebSphere Enterprise applications**.

b. Select the 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 **mfp.topology.clustermode** to `Farm`.
   • Set **mfp.admin.serverid** to the identifier that you chose for this farm member. This identifier must be unique across all farm members.
   • Set **mfp.admin.jmx.user** to a user name that has access to the SOAP connector.
   • Set **mfp.admin.jmx.pwd** to the password of the user as declared in **mfp.admin.jmx.user**.

- Set **mfp.admin.jmx.port** to the value of the SOAP port.

e. Verify that **mfp.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 between all members of the farm. 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"/>
     ```

     Lastly, 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 public certificates of the other servers in the farm into the truststore that is referenced by the `server.xml` configuration file of the server.

      The tutorial ("Installing MobileFirst Server in graphical mode" on page 6-5) provides you the instructions to exchange the certificates between two Liberty servers in a farm. For more information, see step 5 on page 6-21 of "Creating a farm of two Liberty servers that run MobileFirst Server" on page 6-19 section.

   b. Restart each instance of WebSphere Application Server Liberty profile to make the security configuration take effect.

      The following steps are needed for single sign-on (SSO) to work.

   c. Start one member of the farm. In the default LTPA configuration, after the Liberty server starts successfully, it generates an LTPA keystore as *`${wlp.user.dir}`*`/servers/`*`server_name`*`/resources/security/ltpa.keys`.

   d. Copy the `ltpa.keys` file to the *`${wlp.user.dir}`*`/servers/`*`server_name`*`/resources/security` directory of each farm member to replicate the LTPA keystores across the farm members.

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.

Verifying a farm configuration:

You can check the status of the farm members from MobileFirst Operations Console.

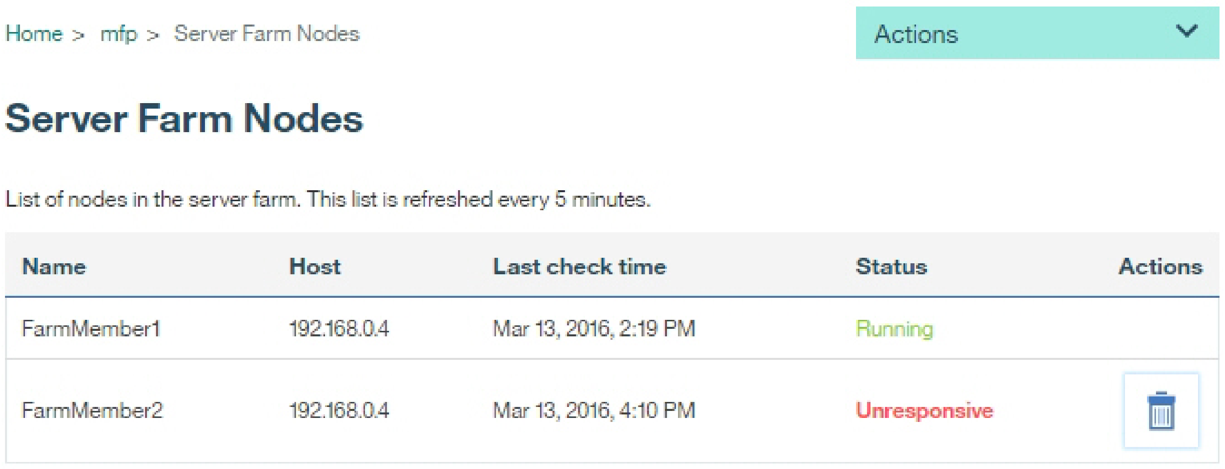
About this task

The purpose of this task is to check the status of the farm members and verify whether a farm is configured properly.

Procedure

1. Start all the servers of the farm.
2. Access MobileFirst Operations Console.

   For example, `http://`*`server_name`*`:`*`port`*`/mfpconsole`, or `https://`*`hostname`*`:`*`secure_port`*`/mfpconsole` in HTTPS. In the console sidebar, an extra menu that is labeled as **Server Farm Nodes** appears.
3. Click **Server Farm Nodes** to access the list of registered farm members and their status. In the following example, the node that is identified as **FarmMember2** is considered to be down, which indicates that this server has probably failed and requires some maintenance.



Server Farm Nodes

List of nodes in the server farm. This list is refreshed every 5 minutes.

Name	Host	Last check time	Status	Actions
FarmMember1	192.168.0.4	Mar 13, 2016, 2:19 PM	Running	
FarmMember2	192.168.0.4	Mar 13, 2016, 4:10 PM	Unresponsive	🗑

Figure 6-7. List of server farm nodes

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.

Registration and monitoring servers as farm nodes

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 periodically send a heartbeat to MobileFirst administration services at a specified rate. If the MobileFirst administration service registers that too long a time has elapsed since a farm member sent a heartbeat, the farm member is considered to be down.

Farm members that are considered to be down do not serve any more requests to mobile applications.

Having one or more nodes down does not prevent the other farm members from correctly serving requests to mobile applications nor from accepting new management operations that are triggered through the 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:

- **`mfp.admin.farm.heartbeat`**
- **`mfp.admin.farm.missed.heartbeats.timeout`**

For more information about JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

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.

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 6-159. 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:

The topologies that are supported by token licensing is listed at "Constraints on MobileFirst Server administration service, MobileFirst Server live update service and MobileFirst runtime" on page 6-84.

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 6-151.

After MobileFirst Server is installed, you must manually configure the server for token licensing. For more information, see the following topics in this section.

The MobileFirst Server is not functional before you complete this manual configuration. 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

After you install and configure IBM MobileFirst Platform Server for token licensing, the server validates licenses during various scenarios. For more information about the retrieval of tokens during operations, see "Token license validation" on page 10-83.

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 "Setting the application license information" on page 10-80.

Installation overview for token licensing

The installation process overview for the use of IBM MobileFirst Platform Foundation with token licensing enabled

Before you begin

Review "Planning for the use of token licensing" on page 6-150.

About this task

If you intend to use token licensing with IBM MobileFirst Platform Foundation, make sure that you go through the following preliminary steps in this order.

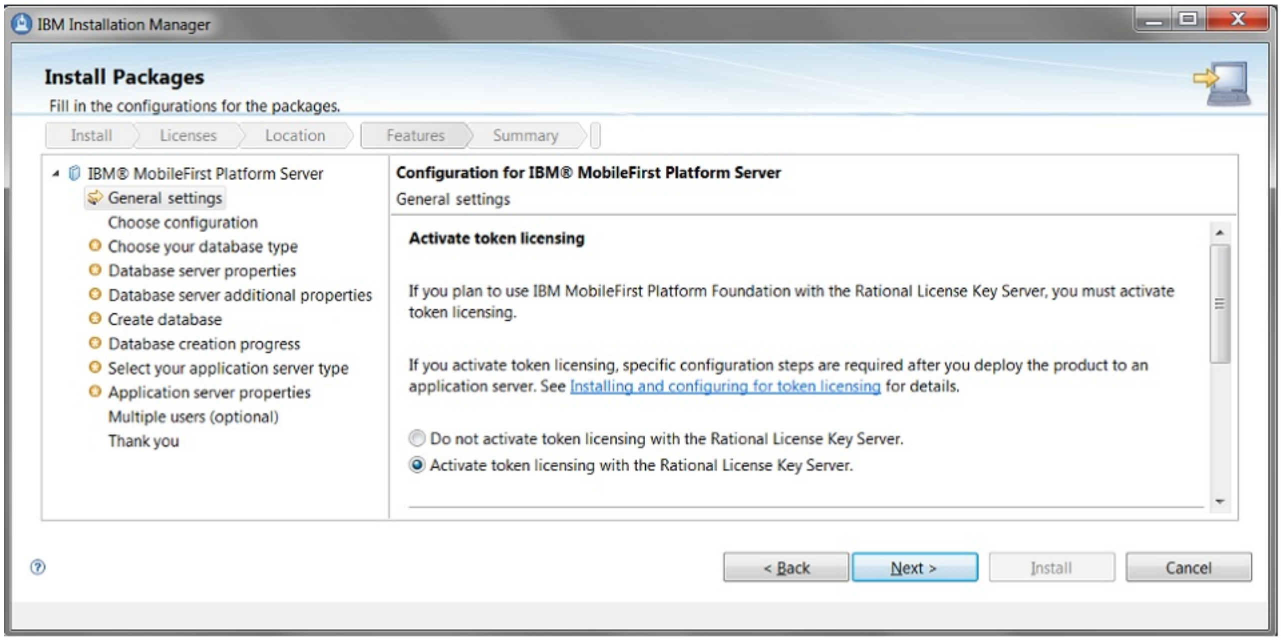
Important:

- Your choice about token licensing (activating it or not) as part of an installation that supports token licensing cannot be modified. If later you need to change the token licensing option, you must uninstall IBM MobileFirst Platform Foundation and reinstall it.

Procedure

1. Activate token licensing when you run IBM Installation Manager to install IBM MobileFirst Platform Foundation.

   Graphic mode installation
      If you install the product in graphic mode, select **Activate token licensing with the Rational License Key Server** option in the **General settings** panel during the installation.


IBM Installation Manager

Install Packages

Fill in the configurations for the packages.

Install › Licenses › Location › Features › Summary

- IBM® MobileFirst Platform Server
  - General settings
  - Choose configuration
  - Choose your database type
  - Database server properties
  - Database server additional properties
  - Create database
  - Database creation progress
  - Select your application server type
  - Application server properties
  - Multiple users (optional)
  - Thank you

Configuration for IBM® MobileFirst Platform Server

General settings

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 token licensing for details.

○ Do not activate token licensing with the Rational License Key Server.

[✓ Activate token licensing with the Rational License Key Server.]

< Back | Next > | Install | Cancel

Figure 6-8. Activating token licensing during the installation of the product.

For more information about running IBM Installation Manager, see "Running IBM Installation Manager" on page 6-40.

Command line 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 example, you can use:

```
imcl install com.ibm.mobilefirst.foundation.server -repositories
mfp_repository_dir/MobileFirst_Platform_Server/disk1 -properties
user.appserver.selection2=none,user.database.selection2=none,user.database.prein
-acceptLicense
```

For more information about installing MobileFirst Server in command line mode, see "Installing by running IBM Installation Manager in command line" on page 6-45.

2. Deploy the MobileFirst Server to an application server after the product installation is complete. For more information, see "Installing MobileFirst Server to an application server" on page 6-100.
3. Configure MobileFirst Server for token licensing. The steps depend on your application server.
   - For WebSphere Application Server Liberty profile, see "Connecting MobileFirst Server installed on WebSphere Application Server Liberty profile to the Rational License Key Server" on page 6-154
   - For Apache Tomcat, see "Connecting MobileFirst Server installed on Apache Tomcat to the Rational License Key Server" on page 6-153
   - For WebSphere Application Server full profile, see "Connecting MobileFirst Server installed on WebSphere Application Server to the Rational License Key Server" on page 6-156.

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.
- Make sure that the license keys for IBM MobileFirst Platform Foundation are generated . For more information about generating and managing your license keys with IBM® Rational License Key Center, see IBM Support - Licensing and Obtaining license keys with IBM Rational License Key Center.
- 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 6-151.

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`*`/MobileFirstServer/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`*`/MobileFirstServer/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`*`/MobileFirstServer/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="mfp.admin.license.key.server.host" value="rlks_hostname" type="java.lang.String" override="false"/>
<Environment name="mfp.admin.license.key.server.port" value="rlks_port" type="java.lang.String" override="false"/>
```

   - *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 about the JNDI properties, 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 6-153 for each node of your server farm where the MobileFirst Server administration service is running. For more information about server farm, see "Server farm topology" on page 6-88 and "Installing a server farm" on page 6-139.

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.
- Make sure that the license keys for IBM MobileFirst Platform Foundation are generated . For more information about generating and managing your license keys with IBM® Rational License Key Center, see IBM Support - Licensing and Obtaining license keys with IBM Rational License Key Center.
- 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 6-151.

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*/MobileFirstServer/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*/MobileFirstServer/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 ${*shared.resource.dir*}/rcllib. The *${shared.resource.dir}* directory is usually in usr/shared/resources, where usr is the directory that also contains the usr/servers directory. 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 (Access is denied).
```

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*/MobileFirstServer/tokenLibs directory.

   Note: The Java virtual machine (JVM) of Liberty profile must have the possibility 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="mfpadmin" name="mfpadmin" location="mfp-admin-service.war" type="war">
  [...]
  <!- Declare the shared library as an attribute commonLibraryRef to
      the class loader of the application. -->
  <classloader delegation="parentLast" commonLibraryRef="RCLLibrary">
  </classloader>
</application>
```

   If you are using Oracle as database, then the server.xml will already have the following class loader:

```
<classloader delegation="parentLast" commonLibraryRef="MobileFirst/JDBC/oracle">
</classloader>
```

   You also need to append Rational Common Licensing library as common library to the Oracle library as follows:

```
<classloader delegation="parentLast"
    commonLibraryRef="MobileFirst/JDBC/oracle,RCLLibrary">
</classloader>
```

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.

The *${wlp.user.dir}* directory is usually in `liberty_install_dir/usr` and contains the `servers` directory. However, it's location can be customized. For more information, see Customizing the Liberty environment

2. Configure MobileFirst Server to access Rational License Key Server.

   In the `${wlp.user.dir}/servers/server_name/server.xml` file, add these JNDI configuration lines.

```
<jndiEntry jndiName="mfp.admin.license.key.server.host" value="rlks_hostname"/>
<jndiEntry jndiName="mfp.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 about the JNDI properties, 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 “Server farm topology” on page 6-88 and “Installing a server farm” on page 6-139.

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.
- Make sure that the license keys for IBM MobileFirst Platform Foundation are generated . For more information about generating and managing your license keys with IBM® Rational License Key Center, see IBM Support - Licensing and Obtaining license keys with IBM Rational License Key Center.
- 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 6-151.

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*/MobileFirstServer/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*/MobileFirstServer/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*/MobileFirstServer/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.

      Note: The native library path for this library is set in step 3 in the *ld.library.path* property of the server's Java virtual machine.

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 the 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: `MobileFirst_Administration_Service`.
   b. In **Web Module Properties** section, click **Environment entries for web modules**.
   c. Enter the values for **`mfp.admin.license.key.server.host`** and **`mfp.admin.license.key.server.port`**.
      - **`mfp.admin.license.key.server.host`** is the host name of the Rational License Key Server.
      - **`mfp.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 6-157. 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.

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 6-22. Supported Operating System, Operating System version, and hardware architecture.

Operating System	Operating System version	Hardware architecture
AIX	7.1	POWER8® (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.

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 service after you complete "Installing and configuring for token licensing" on page 6-150, 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 6-161
- "The Rational License Key Server connection is not configured" on page 6-162
- "The Rational License Key Server is not accessible" on page 6-162
- "Failed to initialize Rational Common Licensing API" on page 6-163
- "Insufficient token licenses" on page 6-163
- "Invalid rcl_ibmratl.jar file" on page 6-163

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 "mfp.admin.license.key.server.host" and
"mfp.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 (**`mfp.admin.license.key.server.host`** and **`mfp.admin.license.key.server.port`**) is not set in the MobileFirst Server administration service 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 6-153
  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 6-154
  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 6-156
  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.

Configuring MobileFirst Server

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

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 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 6-23. MobileFirst runtime endpoints

API URL under `<runtime context root>/api/`	Description	Suggested for whitelist?
`/adapterdoc/*`	Return the adapter's Swagger documentation for the named adapter	No. Used only internally by the administrator and the developers
`/adapters/*`	Adapters serving	Yes
`/az/v1/authorization/*`	Authorize the client to access a specific scope	Yes
`/az/v1/introspection`	Introspect the client's access token	No. This API is for confidential clients only.
`/az/v1/token`	Generate an access token for the client	Yes
`/clientLogProfile/*`	Get client log profile	Yes
`/directupdate/*`	Get Direct Update .zip file	Yes, if you plan to use Direct Update
`/loguploader`	Upload client logs to server	Yes

Table 6-23. MobileFirst runtime endpoints (continued)

API URL under `<runtime context root>/api/`	Description	Suggested for whitelist?
`/preauth/v1/heartbeat`	Accept heartbeat from the client and note the last activity time	Yes
`/preauth/v1/logout`	Log out from a security check	Yes
`/preauth/v1/preauthorize`	Map and execute security checks for a specific scope	Yes
`/reach`	The server is reachable	No, for internal use only
`/registration/v1/clients/*`	Registration-service clients API	No. This API is for confidential clients only.
`/registration/v1/self/*`	Registration-service client self-registration API	Yes

Table 6-24. MobileFirst admin endpoints

API URL under `<admin context root>`	Description	Suggested for whitelist?
`/management-apis/2.0/*`	All the REST APIs of MobileFirst administration service.	Yes. If the client accessing the API is not behind the firewall where the MobileFirst Server is running. No. If the client that accesses the API and the MobileFirst Server are both running behind the firewall.

Configuring MobileFirst Server to enable TLS V1.2

For MobileFirst Server to communicate with devices that support only Transport Layer Security v1.2 (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 it passes the packets to the application server, you must enable TLS V1.2 support on your reverse proxy. If you use IBM HTTP Server as your reverse proxy, see Securing IBM HTTP Server for instructions.
- If MobileFirst Server communicates directly with devices, the steps to enable TLS V1.2 depend on whether your application serveris Apache Tomcat, WebSphere Application Server Liberty profile, or 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:

- Oracle JRE 1.7.0_75 or later
- Oracle JRE 1.8.0_31 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:
     - Oracle JRE 1.7.0_75 or later
     - Oracle JRE 1.8.0_31 or later
2. If you use an IBM Java SDK, edit the `server.xml` file.
   a. Add the following line:

      ```
      <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" sslProtocol="SSL_TLSv2"/>
      ```
   b. Add the `sslProtocol="SSL_TLSv2"` attribute to all existing `<ssl>` elements.

WebSphere Application Server full profile

Procedure

1. Confirm that the Java Runtime Environment (JRE) supports TLS V1.2.

   Ensure that your IBM Java SDK is patched for the POODLE vulnerability. You can find the minimum IBM Java SDK versions that contain the patch for your version of WebSphere Application Server in Security Bulletin: Vulnerability in SSLv3 affects IBM WebSphere Application Server (CVE-2014-3566).

   Note: You can use the versions that are listed in the security bulletin or later versions.
2. Log in to WebSphere Application Server administrative console, and click **Security** > **SSL certificate and key management** > **SSL configurations**.
3. For each SSL configuration listed, modify the configuration to enable TLS V1.2.
   a. Select an SSL configuration and then, under **Additional Properties**, click **Quality of protections (QoP) settings**.
   b. From the **Protocol** list, select `SSL_TLSv2`.
   c. Click **Apply** and then save the changes.

Configuring user authentication for MobileFirst Server administration

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

MobileFirst Server administration requires user authentication.

Important: If you use stand-alone WebSphere Application Server full profile, use an authentication method other than the simple WebSphere authentication method (SWAM) in global security. You can use lightweight third-party authentication (LTPA). If you use SWAM, you might experience unexpected authentication failures.

You must configure 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:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpoperator`**
- **`mfpmonitor`**

You must map the roles to the corresponding sets of users. The **`mfpmonitor`** role can view data but cannot change any data. The following tables list MobileFirst roles and functions for production servers.

Table 6-25. Deployment

	Administrator	**Deployer**	**Operator**	**Monitor**
Java EE security role.	**`mfpadmin`**	**`mfpdeployer`**	**`mfpoperator`**	**`mfpmonitor`**
Deploy an application.	Yes	Yes	No	No
Deploy an adapter.	Yes	Yes	No	No

Table 6-26. MobileFirst Server management

	Administrator	**Deployer**	**Operator**	**Monitor**
Java EE security role.	**`mfpadmin`**	**`mfpdeployer`**	**`mfpoperator`**	**`mfpmonitor`**
Configure runtime settings.	Yes	Yes	No	No

Table 6-27. Application management

	Administrator	**Deployer**	**Operator**	**Monitor**
Java EE security role.	**`mfpadmin`**	**`mfpdeployer`**	**`mfpoperator`**	**`mfpmonitor`**
Upload new MobileFirst application.	Yes	Yes	No	No
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

Basically, all roles can issue GET requests, the **`mfpadmin`**, **`mfpdeployer`**, and **`mfpmonitor`** roles can also issue POST and PUT requests, and the **`mfpadmin`** and **`mfpdeployer`** roles can also issue DELETE requests.

Table 6-28. Requests related to push notifications

	Administrator	Deployer	Operator	Monitor
Java EE security role.	`mfpadmin`	`mfpdeployer`	`mfpoperator`	`mfpmonitor`
GET requests • Get a list of all the devices that use push notification for an application • Get the details of a specific device • Get the list of subscriptions • Get the subscription information that is associated with a subscription ID. • Get the details of a GCM configuration • Get the details of an APNS configuration • Get the list of tags that are defined for the application • Get details of a specific tag	Yes	Yes	Yes	Yes
POST and PUT requests • Register an app with push notification • Update a push device registration • Create a subscription • Add or update a GCM configuration • Add or update an APNS configuration • Submit notifications to a device • Create or update a tag	Yes	Yes	Yes	No
DELETE requests • Delete the registration of a device to push notification • Delete a subscription • Unsubscribe a device from a tag • Delete a GCM configuration • Delete an APNS configuration • Delete a tag	Yes	Yes	No	No

Table 6-29. Disabling

	Administrator	Deployer	Operator	Monitor
Java EE security role.	`mfpadmin`	`mfpdeployer`	`mfpoperator`	`mfpmonitor`
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
Disable a specific application, marking the state as disabled so that access from the specific application on that device is blocked.	Yes	Yes	Yes	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 depends on 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 **`mfpadmin`**, **`mfpdeployer`**, **`mfpmonitor`**, and **`mfpoperator`** 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 **MobileFirst_Administration_Service**.
   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 **MobileFirst_Administration_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 **`mfpadmin`**, **`mfpdeployer`**, **`mfpmonitor`**, and **`mfpoperator`** 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: **`mfpadmin`**, **`mfpdeployer`**, **`mfpmonitor`**, and **`mfpoperator`**. Map the roles to the appropriate user group name, in this example: **`mfpadmingroup`**, **`mfpdeployergroup`**, **`mfpmonitorgroup`**, or **`mfpoperatorgroup`**. 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="mfpadmin">
     <group name="mfpadmingroup"/>
   </security-role>
   <security-role name="mfpdeployer">
     <group name="mfpdeployergroup"/>
   </security-role>
   <security-role name="mfpmonitor">
     <group name="mfpmonitorgroup"/>
   </security-role>
   <security-role name="mfpoperator>
     <group name="mfpoperatorgroup"/>
   </security-role>

   <basicRegistry id="mfpadmin">
     <user name="admin" password="admin"/>
     <user name="guest" password="guest"/>
     <user name="demo" password="demo"/>
     <group name="mfpadmingroup">
       <member name="guest"/>
       <member name="demo"/>
     </group>
     <group name="mfpdeployergroup">
       <member name="admin" id="admin"/>
     </group>
     <group name="mfpmonitorgroup"/>
     <group name="mfpoperatorgroup"/>
   </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="MFPADMIN" jndiName="mfpadmin/jdbc/mfpAdminDS" 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="mfpadmin"/>
   <role rolename="mfpmonitor"/>
   <role rolename="mfpdeployer"/>
   <role rolename="mfpoperator"/>
   ```

2. Add roles to the selected users, for example:

   ```
   <user name="admin" password="admin" roles="mfpadmin"/>
   ```

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

List of JNDI properties of the MobileFirst Server web applications

Configure the JNDI properties for the MobileFirst Server web applications that are deployed to the application server.

Setting up JNDI properties for MobileFirst Server web applications

Set up JNDI properties to configure the MobileFirst Server web applications that are deployed to the application server.

About this task

JNDI environment entries cover all the properties that you can set in a production environment. For details about specific JNDI entries, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174, "List of JNDI properties for MobileFirst Server live update service" on page 6-182, "List of JNDI properties for MobileFirst runtime" on page 6-183, and "List of JNDI properties for MobileFirst Server push service" on page 6-186.

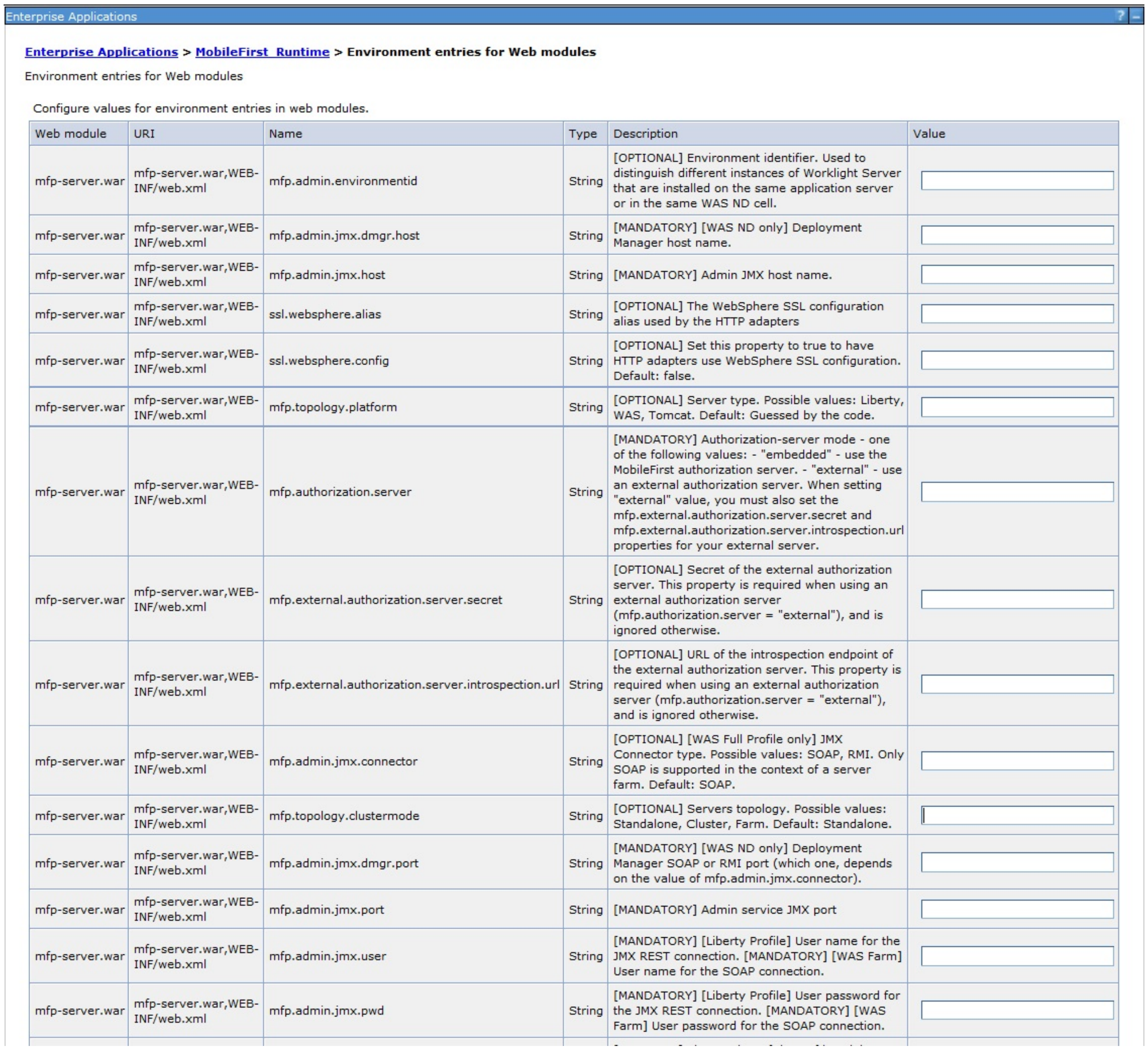
Procedure

Set the JNDI environment entries in one of the following ways:

- 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 WebSphere Application Server administration console, go to **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* > **Environment entries for Web modules**
    2. In the **Value** fields, enter values that are appropriate to your server environment.



Enterprise Applications > **MobileFirst_Runtime** > **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
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.environmentid	String	[OPTIONAL] Environment identifier. Used to distinguish different instances of Worklight Server that are installed on the same application server or in the same WAS ND cell.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.dmgr.host	String	[MANDATORY] [WAS ND only] Deployment Manager host name.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.host	String	[MANDATORY] Admin JMX host name.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	ssl.websphere.alias	String	[OPTIONAL] The WebSphere SSL configuration alias used by the HTTP adapters	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	ssl.websphere.config	String	[OPTIONAL] Set this property to true to have HTTP adapters use WebSphere SSL configuration. Default: false.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.topology.platform	String	[OPTIONAL] Server type. Possible values: Liberty, WAS, Tomcat. Default: Guessed by the code.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.authorization.server	String	[MANDATORY] Authorization-server mode - one of the following values: - "embedded" - use the MobileFirst authorization server. - "external" - use an external authorization server. When setting "external" value, you must also set the mfp.external.authorization.server.secret and mfp.external.authorization.server.introspection.url properties for your external server.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.external.authorization.server.secret	String	[OPTIONAL] Secret of the external authorization server. This property is required when using an external authorization server (mfp.authorization.server = "external"), and is ignored otherwise.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.external.authorization.server.introspection.url	String	[OPTIONAL] URL of the introspection endpoint of the external authorization server. This property is required when using an external authorization server (mfp.authorization.server = "external"), and is ignored otherwise.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.connector	String	[OPTIONAL] [WAS Full Profile only] JMX Connector type. Possible values: SOAP, RMI. Only SOAP is supported in the context of a server farm. Default: SOAP.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.topology.clustermode	String	[OPTIONAL] Servers topology. Possible values: Standalone, Cluster, Farm. Default: Standalone.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.dmgr.port	String	[MANDATORY] [WAS ND only] Deployment Manager SOAP or RMI port (which one, depends on the value of mfp.admin.jmx.connector).	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.port	String	[MANDATORY] Admin service JMX port	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.user	String	[MANDATORY] [Liberty Profile] User name for the JMX REST connection. [MANDATORY] [WAS Farm] User name for the SOAP connection.	
mfp-server.war	mfp-server.war,WEB-INF/web.xml	mfp.admin.jmx.pwd	String	[MANDATORY] [Liberty Profile] User password for the JMX REST connection. [MANDATORY] [WAS Farm] User password for the SOAP connection.	

Figure 6-9. Setting JNDI environment entries on WebSphere Application Server

–

WebSphere Application Server Liberty:

1. In *liberty_install_dir*/usr/servers/*serverName*, edit the server.xml file, and declare the JNDI properties as follows:

```
<application id="app_context_root" name="app_context_root" location="app_war_name.war"
  type="war"> ...
</application>
<jndiEntry jndiName="app_context_root/JNDI_property_name" value="JNDI_property_value" />
```

The context root (in the previous example: *app_context_root*) 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.

Note: Some properties are defined globally on WebSphere Application Server Liberty, without prefixing the property name by the context root. For a list of these properties, see "Global JNDI entries" on page 6-116.

For all other JNDI properties, the names must be prefixed with the context root of the application:

- For the MobileFirst Administration Service application, the MobileFirst Operations Console and MobileFirst runtime, you can define the context root as you want. However, by default it is `/mfpadmin` for MobileFirst Administration Service, `/mfpconsole` for MobileFirst Operations Console, and `/mfp` for MobileFirst runtime.
- For the live update service, the context root must be `/<adminContextRoot>config`. For example, if the context root of the administration service is `/mfpadmin`, then the context root of the live update service must be `/mfpadminconfig`.
- For the push service, you must define the context root as `/imfpush`. Otherwise, the client devices cannot connect to it as the context root is hardcoded in the SDK.

For example:

```
<application id="mfpadmin" name="mfpadmin" location="mfp-admin-service.war"
  type="war"> ...
</application>
<jndiEntry jndiName="mfpadmin/mfp.admin.actions.prepareTimeout" value = "2400000" />
```

– Apache Tomcat:

1. In *tomcat_install_dir*/conf, edit the server.xml file, and declare the JNDI properties as follows:

```
<Context docBase="app_context_root" path="/app_context_root">
  <Environment name="JNDI_property_name" override="false"
    type="java.lang.String" value="JNDI_property_value"/>
</Context>
```

- The context path prefix is not needed because the JNDI entries are defined inside the `<Context>` element of an application.
- `override="false"` is mandatory.
- The `type` attribute is always `java.lang.String`, unless specified differently for the property.

For example:

```
<Context docBase="app_context_root" path="/app_context_root">
  <Environment name="mfp.admin.actions.prepareTimeout" override="false"
    type="java.lang.String" value="2400000"/>
</Context>
```

• If you install with Ant tasks, you can also set the values of the JNDI properties at installation time.

In *mfp_install_dir*/MobileFirstServer/configuration-samples, edit the configuration XML file for the Ant tasks, and declare the values for the JNDI properties by using the `property` element inside the following tags:

– `<installmobilefirstadmin>`, for MobileFirst Server administration, MobileFirst Operations Console, and live update services. For more information, see "Ant tasks for installation of MobileFirst Operations Console, MobileFirst Server artifacts, MobileFirst Server administration, and live update services" on page 6-274.

– `<installmobilefirstruntime>`, for MobileFirst runtime configuration properties. For more information, see "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

– `<installmobilefirstpush>`, for configuration of the push service. For more information, see "Ant tasks for installation of MobileFirst Server push service" on page 6-287.

For example:

```
<installmobilefirstadmin ..>
   <property name = "mfp.admin.actions.prepareTimeout" value = "2400000" />
</installmobilefirstadmin>
```

List of JNDI properties for MobileFirst Server administration service

When you configure MobileFirst Server administration service 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 service

The following properties can be set on the administration service web application `mfp-admin-service.war`.

Table 6-30. JNDI properties for administration service: JMX.

Property	Optional or mandatory	Description	Restrictions
mfp.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.
mfp.admin.jmx.host	Optional	Host name for the JMX REST connection.	Liberty profile only.
mfp.admin.jmx.port	Optional	Port for the JMX REST connection.	Liberty profile only.
mfp.admin.jmx.user	Mandatory for the Liberty profile and for WebSphere Application Server farm, optional otherwise	User name for the JMX REST connection.	WebSphere Application Server Liberty profile: The user name for the JMX REST connection. WebSphere Application Server farm: the user name for the SOAP connection. WebSphere Application Server Network Deployment: the user name of the WebSphere administrator if the virtual host mapped to the MobileFirst server administration application is not the default host. Liberty collective: the user name of the controller administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the Liberty controller.

Table 6-30. JNDI properties for administration service: JMX (continued).

Property	Optional or mandatory	Description	Restrictions
`mfp.admin.jmx.pwd`	Mandatory for the Liberty profile and for WebSphere Application Server farm, optional otherwise	User password for the JMX REST connection.	WebSphere Application Server Liberty profile: the user password for the JMX REST connection. WebSphere Application Server farm: the user password for the SOAP connection. WebSphere Application Server Network Deployment: the 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. Liberty collective: the password of the controller administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the Liberty controller.
`mfp.admin.rmi.registryPort`	Optional	RMI registry port for the JMX connection through a firewall.	Tomcat only.
`mfp.admin.rmi.serverPort`	Optional	RMI server port for the JMX connection through a firewall.	Tomcat only.
`mfp.admin.jmx.dmgr.host`	Mandatory	Deployment manager host name.	WebSphere Application Server Network Deployment only.
`mfp.admin.jmx.dmgr.port`	Mandatory	Deployment manager RMI or SOAP port.	WebSphere Application Server Network Deployment only.

Table 6-31. JNDI properties for administration service: timeout.

Property	Optional or mandatory	Description
`mfp.admin.actions.prepareTimeout`	Optional	Timeout in milliseconds to transfer data from the adminstration 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 6-31. JNDI properties for administration service: timeout (continued).

Property	Optional or mandatory	Description
`mfp.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)
`mfp.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)
`mfp.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 6-32. JNDI properties for administration service: logging.

Property	Optional or mandatory	Description
`mfp.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`.
`mfp.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 6-33. JNDI properties for administration service: proxies.

Property	Optional or mandatory	Description
mfp.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.
mfp.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.
mfp.admin.proxy.scheme	Optional	This property is just an alternative name for **mfp.admin.proxy.protocol**.
mfp.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 6-34. JNDI properties for administration service: topologies.

Property	Optional or mandatory	Description
mfp.admin.audit	Optional.	Set this property to false to disable the audit feature of the MobileFirst Operations Console. The default value is true.
mfp.admin.environmentid	Optional.	The 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 service, which console, and which runtimes belong to the same installation. The administration service manages only the runtimes that have the same environment identifier.
mfp.admin.serverid	Mandatory for server farms and Liberty collective, optional otherwise.	Server farm: the server identifier. Must be different for each server in the farm. Liberty collective: the value must be controller.

Table 6-34. JNDI properties for administration service: topologies (continued).

Property	Optional or mandatory	Description
mfp.admin.hsts	Optional.	Set to true to enable HTTP Strict Transport Security according to RFC 6797.
mfp.topology.platform	Optional	Server type. Valid values: • `Liberty` • `WAS` • `Tomcat` If you do not set the value, the application tries to guess the server type.
mfp.topology.clustermode	Optional	In addition to the server type, specify here the server topology. Valid values: • `Standalone` • `Cluster` • `Farm` The default value is `Standalone`.
mfp.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 6-149.
mfp.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 6-149.
mfp.admin.farm.reinitialize	Optional	A Boolean value (true or false) for re-registering or re-initializing the farm member.
mfp.swagger.ui.url	Optional	This property defines the URL of the Swagger user interface to be displayed in the administration console.

Table 6-35. JNDI properties for administration service: relational database.

Property	Optional or mandatory	Description
mfp.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/mfpAdminDS`.
mfp.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 **mfp.admin.db.jndi.name** property is not defined in the application server configuration.
mfp.admin.db.openjpa.ConnectionURL	Optional Conditionally mandatory	The URL for the database connection. Mandatory only when the data source that is specified by the **mfp.admin.db.jndi.name** property is not defined in the application server configuration.
mfp.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 **mfp.admin.db.jndi.name** property is not defined in the application server configuration.
mfp.admin.db.openjpa.ConnectionPassword	Optional Conditionally mandatory	The password for the database connection. Mandatory only when the data source that is specified by the **mfp.admin.db.jndi.name** property is not defined in the application server configuration.
mfp.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.
mfp.admin.db.type	Optional	This property defines the type of database. The default value is inferred from the connection URL.

Table 6-36. JNDI properties for administration service: licensing.

Property	Optional or mandatory	Description
mfp.admin.license.key.server.host	• Optional for perpetual licenses • Mandatory for token licenses	Host name of the Rational License Key Server.
mfp.admin.license.key.server.port	• Optional for perpetual licenses • Mandatory for token licenses	Port number of the Rational License Key Server.

Table 6-37. JNDI properties for administration service: JNDI configurations

Property	Optional or mandatory	Description
`mfp.jndi.configuration`	Optional	The name of the JNDI configuration if the JNDI properties (except this one) must be read from a property file that is injected into the WAR file. If you do not set this property, JNDI properties are not read from a property file.
`mfp.jndi.file`	Optional	The name of the file that contains the JNDI configuration if the JNDI properties (except this one) must be read from a file installed in the web server. If you do not set this property, JNDI properties are not read from a property file.

The administration service uses a live update service as an auxiliary facility to store various configurations. Use these properties to configure how to reach the live update service.

Table 6-38. JNDI properties for administration service: live update service

Property	Optional or mandatory	Description
`mfp.config.service.url`	Optional	The URL of the live update service. The default URL is derived from the URL of administration service by adding `config` to the context root of the administration service.
`mfp.config.service.user`	Mandatory	The user name that is used to access the live update service. In a server farm topology, the user name must be the same for all the members of the farm.
`mfp.config.service.password`	Mandatory	The password that is used to access the live update service. In a server farm topology, the password must be the same for all the members of the farm.
`mfp.config.service.schema`	Optional	The name of the schema that is used by the live update service.

The administration service uses a push service as an auxiliary facility to store various push settings. Use these properties to configure how to reach the push service. Because the push service is protected by the OAuth security model, you must set various properties to enable confidential clients in OAuth.

Table 6-39.

Property	Optional or mandatory	Description
`mfp.admin.push.url`	Optional	The URL of the push service. If the property is not specified, the push service is considered disabled. If the property is not properly set, the administration service cannot contact the push service and the administration of push services in MobileFirst Operations Console does not work.
`mfp.admin.authorization.server.url`	Optional	The URL of the OAuth authorization server that is used by the push service. The default URL is derived from the URL of the administration service by changing the context root to the context root of the first installed runtime. If you install multiple runtimes, it is best to set the property. If the property is not set properly, the administration service cannot contact the push service and the administration of push services in MobileFirst Operations Console does not work.
`mfp.push.authorization.client.id`	Optional, conditionally mandatory	The identifier of the confidential client that handles OAuth authorization for the push service. Mandatory only if the **`mfp.admin.push.url`** property is specified.
`mfp.push.authorization.client.secret`	Optional, conditionally mandatory	The secret of the confidential client that handles OAuth authorization for the push service. Mandatory only if the **`mfp.admin.push.url`** property is specified.
`mfp.admin.authorization.client.id`	Optional, conditionally mandatory	The identifier of the confidential client that handles OAuth authorization for the administration service. Mandatory only if the **`mfp.admin.push.url`** property is specified.
`mfp.push.authorization.client.secret`	Optional, conditionally mandatory	The secret of the confidential client that handles OAuth authorization for the administration service. Mandatory only if the **`mfp.admin.push.url`** property is specified.

JNDI properties for MobileFirst Operations Console

The following properties can be set on the web application (`mfp-admin-ui.war`) of MobileFirst Operations Console.

Table 6-40. JNDI properties for the MobileFirst Operations Console.

Property	Optional or mandatory	Description
mfp.admin.endpoint	Optional	Enables the MobileFirst Operations Console to locate the MobileFirst Server administration REST service. Specify the external address and context root of the `mfp-admin-service.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/mfpadmin`.
mfp.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`.
mfp.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`.
mfp.admin.ui.cors	Optional	The default value is `true`. For more information, see the W3C Cross-Origin Resource Sharing page.
mfp.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 service is not, or conversely. This property takes effect only if the **mfp.admin.ui.cors** property is enabled.

To know how to set those properties, see "Setting up JNDI properties for MobileFirst Server web applications" on page 6-171.

Related reference:

JNDI environment entries for MobileFirst runtime
When you configure the MobileFirst Server runtime for your application server, you need to set the optional or mandatory JNDI properties.

List of JNDI properties for MobileFirst Server live update service

When you configure the MobileFirst Server live update service for your application server, you can set the following JNDI properties.

The table lists the JNDI properties for the IBM relational database live update service.

Table 6-41. JNDI properties for the live update service: IBM relational database

Property	Optional or mandatory	Description
mfp.db.relational.queryTimeout	Optional	Timeout for executing a query in RDBMS, in seconds. A value of zero means an infinite timeout. A negative value means the default (no override). In case no value is configured, a default value is used. For more information, see setQueryTimeout.

To know how to set those properties, see "Setting up JNDI properties for MobileFirst Server web applications" on page 6-171.

List of JNDI properties for MobileFirst runtime

When you configure the MobileFirst Server runtime for your application server, you need to set the optional or mandatory JNDI properties.

The following table lists the MobileFirst properties that are always available as JNDI entries:

Table 6-42. MobileFirst properties available as JNDI entries.

Property name	Description
mfp.admin.jmx.dmgr.host	Mandatory. The host name of the deployment manager. WebSphere Application Server Network Deployment only.
mfp.admin.jmx.dmgr.port	Mandatory. The RMI or SOAP port of the deployment manager. WebSphere Application Server Network Deployment only.
mfp.admin.jmx.host	Liberty only. The host name for the JMX REST connection. For Liberty collective, use the host name of the controller.
mfp.admin.jmx.port	Liberty only. The port number for the JMX REST connection. For Liberty collective, the port of the REST connector must be identical to the value of the **httpsPort** attribute that is declared in the `<httpEndpoint>` element. This element is declared in the `server.xml` file of the Liberty controller.
mfp.admin.jmx.user	Optional. WebSphere Application Server farm: the user name of the SOAP connection. Liberty collective: the user name of the controller administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the Liberty controller.
mfp.admin.jmx.pwd	Optional. WebSphere Application Server farm: the user passsword of the SOAP connection. Liberty collective: the password of the controller administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the Liberty controller.
mfp.admin.serverid	Mandatory for server farms and Liberty collective, optional otherwise. Server farm: the server identifier. Must be different for each server in the farm. Liberty collective: the member identifier. The identifier must be different for each member in the collective. The value `controller` cannot be used as it is reserved for the collective controller.

Table 6-42. MobileFirst properties available as JNDI entries (continued).

Property name	Description
mfp.topology.platform	Optional. The server type. Valid values are: • `Liberty` • `WAS` • `Tomcat` If you do not set the value, the application tries to guess the server type.
mfp.topology.clustermode	Optional. In addition to the server type, specify here the server topology. Valid values: • `Standalone` • `Cluster` • `Farm` The default value is `Standalone`.
mfp.admin.jmx.replica	Optional. For Liberty collective only. Set this property only when the administration components that manage this runtime are deployed in different Liberty controllers (replicas). Endpoint list of the different controller replicas with the following syntax: `replica-1 hostname:replica-1 port, replica-2 hostname:replica-2 port,..., replica-n hostname:replica-n port`
mfp.analytics.console.url	Optional. The URL that is exposed by IBM MobileFirst Analytics that links to the Analytics console. Set this property if you want to access the Analytics console from the MobileFirst Operations Console. For example, `http://<hostname>:<port>/analytics/console`
mfp.analytics.password	The password that is used if the data entry point for the IBM MobileFirst Analytics is protected with basic authentication.
mfp.analytics.url	The URL that is exposed by the IBM MobileFirst Analytics that receives incoming analytics data. For example, `http://<hostname>:<port>/analytics-service/rest`
mfp.analytics.username	The user name that is used if the data entry point for the IBM MobileFirst Analytics is protected with basic authentication.
mfp.device.decommissionProcessingInterval	Defines how often (in seconds) the decommissioning task is executed. Default: `86400`, which is one day.
mfp.device.decommission.when	The number of days of inactivity after which a client device is decommissioned by the device decommissioning task. Default: `90 days`.

Table 6-42. MobileFirst properties available as JNDI entries (continued).

Property name	Description
mfp.device.archiveDecommissioned.when	The number of days of inactivity, after which a client device that has been decommissioned is archived. This task writes the client devices that were decommissioned to an archive file. 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`.
mfp.licenseTracking.enabled	A value that is used to enable or disable device tracking in IBM MobileFirst Platform Foundation. For performance reasons, you can disable device tracking when IBM MobileFirst Platform Foundation runs only Business-to-Consumer (B2C) apps. When device tracking is disabled, the license reports are also disabled and no license metrics are generated. Possible values are `true` (default) and `false`.
mfp.runtime.temp.folder	Defines the runtime temporary files folder. Uses the default temporary folder location of the web container when not set.
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.authorization.server	Authorization-server mode. Can be one of the following mode: • `embedded`: Use the MobileFirst authorization server. • `external`: Use an external authorization server. When setting this value, you must also set the **mfp.external.authorization.server.secret** and **mfp.external.authorization.server.introspection.url** properties for your external server.
mfp.external.authorization.server.secret	Secret of the external authorization server. This property is required when using an external authorization server, meaning **mfp.authorization.server** is set to `external` and is ignored otherwise.
mfp.external.authorization.server.introspection.url	URL of the introspection endpoint of the external authorization server. This property is required when using an external authorization server, meaning **mfp.authorization.server** is set to `external` and is ignored otherwise.
ssl.websphere.config	Used to configure the keystore for an HTTP adapter. When set to `false` (default), instructs the MobileFirst runtime to use the MobileFirst keystore. When set to `true`, instructs the MobileFirst runtime to use the WebSphere SSL configuration. For more information, see WebSphere Application Server SSL configuration and HTTP adapters.

To know how to set those properties, see "Setting up JNDI properties for MobileFirst Server web applications" on page 6-171.

List of JNDI properties for MobileFirst Server push service

When you configure MobileFirst Server push notification for your application server, you need to set the optional or mandatory JNDI properties.

Table 6-43. JNDI properties for Push service.

Property	Optional or mandatory	Description
mfp.push.db.type	Optional	Database type. Possible values: DB, CLOUDANT Default: DB
mfp.push.db.queue.connections	Optional	Number of threads in the thread pool that does the database operation. Default: 3
mfp.push.db.cloudant.url	Optional	The Cloudant account URL. When this property is defined, the Cloudant DB will be directed to this URL.
mfp.push.db.cloudant.dbName	Optional	The name of the database in the Cloudant account. It must start with a lowercase letter and consist only of lowercase letters, digits, and the characters _, $, and -. Default: mfp_push_db
mfp.push.db.cloudant.username	Optional	The user name of the Cloudant account, used to store the database. when this property is not defined, a relational database is used.
mfp.push.db.cloudant.password	Optional	The password of the Cloudant account, used to store the database. This property must be set when mfp.db.cloudant.username is set.
mfp.push.db.cloudant.doc.version	Optional	The Cloudant document version.
mfp.push.db.cloudant.socketTimeout	Optional	A timeout for detecting the loss of a network connection for Cloudant, in milliseconds. A value of zero means an infinite timeout. A negative value means the default (no override). Default. See https://github.com/cloudant/java-cloudant#advanced-configuration.

Table 6-43. JNDI properties for Push service (continued).

Property	Optional or mandatory	Description
`mfp.push.db.cloudant.connectionTimeout`	Optional	A timeout for establishing a network connection for Cloudant, in milliseconds. A value of zero means an infinite timeout. A negative value means the default (no override). Default. See https://github.com/cloudant/java-cloudant#advanced-configuration.
`mfp.push.db.cloudant.maxConnections`	Optional	The Cloudant connector's max connections. Default. See https://github.com/cloudant/java-cloudant#advanced-configuration.
`mfp.push.db.cloudant.ssl.authentication`	Optional	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
`mfp.push.db.cloudant.ssl.configuration`	Optional	[WAS Full Profile only] For HTTPS connections to the Cloudant database: 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.push.db.cloudant.proxyHost`	Optional	Cloudant connector's proxy host. Default: See https://github.com/cloudant/java-cloudant#advanced-configuration.
`mfp.push.db.cloudant.proxyPort`	Optional	Cloudant connector's proxy port. Default: See https://github.com/cloudant/java-cloudant#advanced-configuration.
`mfp.push.services.ext.security`	Optional	The security extension plugin.
`mfp.push.security.endpoint`	Optional	The endpoint URL for the authorization server.

Table 6-43. JNDI properties for Push service (continued).

Property	Optional or mandatory	Description
mfp.push.security.user	Optional	The username to access the authorization server.
mfp.push.security.password	Optional	The password to access the authorization server.
mfp.push.services.ext.analytics	Optional	The analytics extension plugin.
mfp.push.analytics.endpoint	Optional	The endpoint URL for the analytics server.
mfp.push.analytics.user	Optional	The username to access the analytics server.
mfp.push.analytics.password	Optional	The password to access the analytics server.
mfp.push.analytics.events.appCreate	Optional	The analytic event when the application is created. Default: true
mfp.push.analytics.events.appDelete	Optional	The analytic event when the application is deleted. Default: true
mfp.push.analytics.events.deviceRegister	Optional	The analytic event when the device is registered. Default: true
mfp.push.analytics.events.deviceUnregister	Optional	The analytic event when the device is unregistered. Default: true
mfp.push.analytics.events.tagSubscribe	Optional	The analytic event when the device is subscribed to tag. Default: true
mfp.push.analytics.events.tagUnsubscribe	Optional	The analytic event when the device is unsubscribed from tag. Default: true
mfp.push.analytics.events.notificationSendSuccess	Optional	The analytic event when the notification is sent successfully. Default: true

Table 6-43. JNDI properties for Push service (continued).

Property	Optional or mandatory	Description
`mfp.push.analytics.events.notificationSendFailure`	Optional	The analytic event when the notification is failed to send. Default: false
`mfp.push.analytics.events.inactiveDevicePurge`	Optional	The analytic event when the inactive devices are deleted. Default: true
`mfp.push.analytics.events.msgReqAccepted`	Optional	The analytic event when the notification is accepted for delivery. Default: true
`mfp.push.analytics.events.msgDispatchFailed`	Optional	The analytic event when the notification dispatch failed. Default: true
`mfp.push.analytics.events.notificationDispatch`	Optional	The analytic event when the notification is about to be dispatched. Default: true
`mfp.push.internalQueue.maxLength`	Optional	The length of the queue which holds the notification tasks before dispatch. Default: 200000
`mfp.push.gcm.proxy.enabled`	Optional	Shows whether Google GCM must be accessed through a proxy. Default: false
`mfp.push.gcm.proxy.protocol`	Optional	Can be either http or https.
`mfp.push.gcm.proxy.host`	Optional	GCM proxy host. Negative value means default port.
`mfp.push.gcm.proxy.port`	Optional	GCM proxy port. Default: -1
`mfp.push.gcm.proxy.user`	Optional	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.

Table 6-43. JNDI properties for Push service (continued).

Property	Optional or mandatory	Description
`mfp.push.gcm.proxy.password`	Optional	Proxy password, if the proxy requires authentication.
`mfp.push.gcm.connections`	Optional	Push GCM max connections. Default : 10
`mfp.push.apns.proxy.enabled`	Optional	Shows whether APNs must be accessed through a proxy. Default: false
`mfp.push.apns.proxy.type`	Optional	APNs proxy type.
`mfp.push.apns.proxy.host`	Optional	APNs proxy host.
`mfp.push.apns.proxy.port`	Optional	APNs proxy port. Default: -1
`mfp.push.apns.proxy.user`	Optional	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.
`mfp.push.apns.proxy.password`	Optional	Proxy password, if the proxy requires authentication.
`mfp.push.apns.connections`	Optional	Push APNs max connections. Default : 3
`mfp.push.apns.connectionIdleTimeout`	Optional	APNs Idle Connection Timeout. Default : 0

To know how to set those properties, see "Setting up JNDI properties for MobileFirst Server web applications" on page 6-171.

Configuring data sources

Find out some data source configuration details pertaining to the supported databases.

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.

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.

Handling stale connections

Configure your application server to avoid database timeout issues.

A StaleConnectionException is an exception that is generated by the Java application server 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.

Apache Tomcat configuration

MySQL

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 statement closed.
```

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

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_value"/>
```

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 .

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.

WebSphere Application Server full profile configuration

DB2 or Oracle

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.

MySQL

1. Log in to the WebSphere Application Server administration 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**.

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.

Stale data after creating or deleting apps from MobileFirst Operations Console

On a Tomcat 8 application server, if you use a MySQL database, some calls from MobileFirst Operations Console to services return a `404` error.

On a Tomcat 8 application server, if you work with a MySQL database, when you use MobileFirst Operations Console to delete an app, or add a new one, and try to refresh the console a couple of times, you might see stale data. For example, users might see an already deleted app in the list.

To avoid this problem, change the isolation level to `READ_COMMITTED`, either in the data source, or in the database management system.

For the meaning of `READ_COMMITTED`, see the MySQL documentation at `http://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html`.

- To change the isolation level to `READ_COMMITTED` in the data source, modify the `server.xml` Tomcat configuration file: In the `<Resource name="jdbc/mfpAdminDS" .../>` section, add the `defaultTransactionIsolation="READ_COMMITTED"` attribute.
- To change the isolation level to `READ_COMMITTED` globally in the database management system, refer to the SET TRANSACTION Syntax page of the MySQL documentation at `http://dev.mysql.com/doc/refman/5.7/en/set-transaction.html`.

Configuring logging and monitoring mechanisms

IBM MobileFirst Platform Foundation reports errors, warnings, and informational messages into a log file. The underlying logging mechanism varies by application server.

MobileFirst Server

IBM MobileFirst Platform Server (MobileFirst Server for short) uses the standard `java.util.logging` package. By default, all MobileFirst logging goes to the application server log files. You can control MobileFirst Server logging by using the standard tools that are available in each application server. For example, if you want to activate trace logging in WebSphere Application Server 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.ibm.mfp`.

Application Center logs begin with `com.ibm.puremeap`.

For more information about the logging models of each application server, including the location of the log files, see the documentation for the relevant application server, as shown in the following table.

Table 6-44. Documentation for different server platforms.

Application server	Location of documentation
Apache Tomcat	http://tomcat.apache.org/tomcat-7.0-doc/logging.html#Using_java.util.logging_(default)
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

Table 6-44. Documentation for different server platforms (continued).

Application server	Location of documentation
WebSphere Application Server Version 8.5 Liberty profile	http://ibm.biz/knowctr#SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/rwlp_logging.html?cp=SSEQTP_8.5.5%2F1-16-0-0

Log level mappings

MobileFirst Server uses the `java.util.logging` API. 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 IBM Knowledge Center. The URLs are listed in the table in the “MobileFirst Server” on page 6-194 section of this page.

Back-end connectivity

To enable trace to monitor back-end connectivity, see the documentation for your specific application server platform in the table of section “MobileFirst Server” on page 6-194 of this page. Use the `com.ibm.mfp.server.js.adapter` package and set the log level to `FINEST`.

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. You can disable the audit log by setting the JNDI property **`mfp.admin.audit`** to `false` on the web application of the MobileFirst administration service (`mfp-admin-service.war`).

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.

Login and authentication issues

To diagnose login and authentication issues, enable the package `com.ibm.mfp.server.security` for trace and set the log level to `FINEST`.

Configuring license tracking

License tracking is enabled by default but can optionally be configured.

About this task

License tracking is enabled by default. Read the following topics to learn how you can configure license tracking. For more information about license tracking, see "License tracking" on page 10-80

Configuring license tracking for client device and addressable device

License tracking for client devices and addressable device is enabled by default. License reports are available in the MobileFirst Operations Console. You can specify the following JNDI properties to change the default settings for license tracking.

About this task

Note: License tracking for client devices and addressable device is enabled by default. Configure license tracking only if you need to change the default settings.

Note: If you have a contract that defines the use of token licensing, see also "Installing and configuring for token licensing" on page 6-150.

License tracking for client devices and addressable device is enabled by default. You can specify the following JNDI properties to change the default settings for license tracking.

`mfp.device.decommission.when`
: The number of days of inactivity after which a device is decommissioned by the device decommissioning task. License reports do not count decommissioned devices as active devices. The default value for the property is 90 days. Do not set a value lower than 30 days if your software is licensed by Client Device or by Addressable Device, or license reports might not be sufficient to prove compliance.

`mfp.device.archiveDecommissioned.when`
: A value, in days, that defines when decommissioned devices are placed in an archive file when the decommissioning task is run. The archived 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.

`mfp.device.decommissionProcessingInterval`
: Defines how often (in seconds) the decommissioning task is run. Default: `86400`, which is one day. The decommissioning task performs the following actions:
  - Decommissions inactive devices, based on the **`mfp.device.decommission.when`** setting.
  - Optionally, archives older decommissioned devices, based on the **`mfp.device.archiveDecommissioned.when`** setting.
  - Generates the license tracking report.

`mfp.licenseTracking.enabled`
: A value that is used to enable or disable license tracking in IBM MobileFirst Platform Foundation. By default, license tracking is enabled. For performance reasons, you can disable this flag when IBM MobileFirst Platform Foundation is not licensed by Client Device or by Addressable Device. When device

tracking is disabled, the license reports are also disabled and no license metrics are generated. In that case, only IBM License Metric Tool records for Application count are generated.

For more information about specifying JNDI properties, see "List of JNDI properties for MobileFirst runtime" on page 6-183.

For more information about license tracking, see "License tracking" on page 10-80.

Configuring IBM License Metric Tool log files

IBM MobileFirst Platform Foundation 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 to understand how to configure the location and the maximum size of the generated files.

About this task

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.

For more information about IBM License Metric Tool reports, see "Integration with IBM License Metric Tool" on page 10-84.

WebSphere Application Server SSL configuration and HTTP adapters

By setting a property, you can let HTTP adapters benefit from WebSphere SSL configuration.

By default, HTTP adapters do not use WebSphere SSL by concatenating the Java Runtime Environment (JRE) truststore with the IBM MobileFirst Platform Server keystore, which is described in "Configuring the MobileFirst Server keystore" on page 7-316. See "Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates" on page 10-3.

To have HTTP adapters use the WebSphere SSL configuration, set the **ssl.websphere.config** JNDI property to `true`. The setting has the following effects in order of precedence:

1. Adapters running on WebSphere use the WebSphere keystore and not the IBM MobileFirst Platform Server keystore.
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.

Installing and configuring the MobileFirst Analytics Server

The MobileFirst Analytics Server is delivered as two separate WAR files. For convenience in deploying on WebSphere Application Server or WebSphere Application Server Liberty, MobileFirst Analytics Server is also delivered as an EAR file that contains the two WAR files.

Note: Do not install more than one instance of MobileFirst Analytics Server on a single host machine. For more information about managing your cluster, see the Elasticsearch documentation.

The analytics WAR and EAR files are included with the MobileFirst Server installation. For more information, see "Distribution structure of MobileFirst Server" on page 6-61.

When you deploy the WAR file, the MobileFirst Analytics Console is available at:

```
http://<hostname>:<port>/analytics/console
```

Example:

```
http://localhost:9080/analytics/console
```

For more information about how to install MobileFirst Analytics Server, see "MobileFirst Analytics Server installation guide" on page 11-2.

For more information about how to configure IBM MobileFirst Analytics, see "Configuration guide" on page 11-14.

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 6-233.

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 6-61.

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 6-104.

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 6-40.
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 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 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 USERS;
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 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 6-45. 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 6-203.

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 Server to the cloud" on page 9-1.

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 6-234.

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 6-236.

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 6-237.

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*/MobileFirstServer/mfp-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 6-199, follow steps 1 to 3.
- If your database already exists, you must create only the database tables. Follow steps 4 to 7.

Procedure

If you did not create your database manually, as described in "Optional creation of databases" on page 6-199, 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" on page 6-206.

   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" on page 6-206.

See also:

- "Ant tasks for installation of Application Center" on page 6-304
- "Sample configuration files" on page 6-318

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 6-204.
- 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*/MobileFirstServer/mfp-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.

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 6-199.
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 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 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 6-227.

Configuring the Apache Derby database manually for Application Center

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

Procedure

1. Create the database and the tables within them. This step is described in "Setting up your Apache Derby database manually for Application Center."
2. Configure the application server to use this database setup. Go to one of the following topics:
   - "Configuring Liberty profile for Derby manually for Application Center" on page 6-212
   - "Configuring WebSphere Application Server for Derby manually for Application Center" on page 6-212
   - "Configuring Apache Tomcat for Derby manually for Application Center" on page 6-214

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 2-7.
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.EmbeddedConnectionPoolDataSource40"/>
  <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 6-211.
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 6-227.

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 6-200.
2. Create the tables in the database. This step is described in "Setting up your MySQL database manually for Application Center" on page 6-215.
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 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 6-227.

```
<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 6-201.
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 USE
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 6-222.

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 6-46. 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 6-227

   ```
   <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:

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 6-207, 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:

   ```
   <feature>jdbc-4.0</feature>
   <feature>appSecurity-2.0</feature>
   <feature>servlet-3.0</feature>
   <feature>usr:MFPDecoderFeature-1.0</feature>
   ```

2. Add the following declarations for Application Center:

```
<!-- The directory with binaries of the 'aapt' program, from the Android SDK's
     platform-tools package. -->
<jndiEntry jndiName="android.aapt.dir" value="product_install_dir/ApplicationCenter/tools/android-sdk"/>
<!-- 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">
  </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">
  </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` are example logins that 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 6-236.

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">
...
```

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 6-219.

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 LIBERTY_HOME/wlp/usr/servers/server_name
     ```
   - On Windows systems:

```
mmkdir 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. Copy the password decoder user feature.
   - On UNIX and Linux systems:

```
mkdir -p LIBERTY_HOME/wlp/usr/extension/lib/features
cp product_install_dir/features/com.ibm.websphere.crypto_1.0.0.jar LIBERTY_HOME/wlp/usr/extension/lib/
cp product_install_dir/features/MFPDecoderFeature-1.0.mf LIBERTY_HOME/wlp/usr/extension/lib/features/
```

   - On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\extension\lib
copy /B product_install_dir\features\com.ibm.websphere.crypto_1.0.0.jar
LIBERTY_HOME\wlp\usr\extension\lib\com.ibm.websphere.crypto_1.0.0.jar
mkdir LIBERTY_HOME\wlp\usr\extension\lib\features
copy /B product_install_dir\features\MFPDecoderFeature-1.0.mf
LIBERTY_HOME\wlp\usr\extension\lib\features\MFPDecoderFeature-1.0.mf
```

6. 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 6-236.

Configuring WebSphere Application Server for Application Center manually:

To configure WebSphere Application Server for Application Center manually, you must configure variables, custom properties, and class 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 6-207.
4. Install the Application Center console WAR file.
   a. Depending on your version of WebSphere Application Server, click one of the following options:
      - **Applications** > **New** > **New Enterprise Application**
      - **Applications** > **New Application** > **New Enterprise Application**
   b. Navigate to the 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. Navigate 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 classloader policy of all applications other than MobileFirst applications to **Classes loaded with parent class loader first**.

9. Save the configuration.

10. Configure a JNDI environment entry to indicate the directory with binary files of the `aapt` program, from the Android SDK `platform-tools` package.

    a. Determine a suitable directory for the `aapt` binary files in the WebSphere Application Server installation directory.

       - For a stand-alone server, you can use a directory such as *`WAS_INSTALL_DIR`*`/optionalLibraries/IBM/mobilefirst/android-sdk`.
       - For deployment to a WebSphere Application Server Network Deployment cell, use *`WAS_INSTALL_DIR`*`/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/mobilefirst/android-sdk`.
       - For deployment to a WebSphere Application Server Network Deployment cluster, use *`WAS_INSTALL_DIR`*`/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/clusters/`*`cluster-name`*`/mobilefirst/android-sdk`.
       - For deployment to a WebSphere Application Server Network Deployment node, use *`WAS_INSTALL_DIR`*`/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/mobilefirst/android-sdk`.
       - For deployment to a WebSphere Application Server Network Deployment server, use *`WAS_INSTALL_DIR`*`/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/servers/`*`server-name`*`/mobilefirst/android-sdk`.

b. Copy the *product_install_dir*/ApplicationCenter/tools/android-sdk directory to the directory that you determined in Substep a.
c. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.
d. Configure the environment entry (JNDI property) android.aapt.dir, and set as its value the directory that you determined in Substep a. The *WAS_INSTALL_DIR*/profiles/*profile-name* profile is replaced with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.

Results

You can now access the Application Center at http://<*server*>:<*port*>/appcenterconsole, where *server* is the host name of your server and *port* is the port number (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 6-234.

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 6-207.
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 directory with binaries of the 'aapt' program, from
       the Android SDK's platform-tools package. -->
  <Environment name="android.aapt.dir"
               value="product_install_dir/ApplicationCenter/tools/android-sdk"
               type="java.lang.String" override="false"/>
  <!-- The protocol of the application resources URI.
```

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

     <!-- The 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 6-210
- "Configuring Apache Tomcat for Derby manually for Application Center" on page 6-214
- "Configuring Apache Tomcat for MySQL manually for Application Center" on page 6-217
- "Configuring Apache Tomcat for Oracle manually for Application Center" on page 6-222

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_dir\
     copy /B product_install_dir\ApplicationCenter\console\applicationcenter.war tomcat_install_dir
     ```
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 6-237.

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" on page 6-207, 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:

   ```
   <feature>jdbc-4.0</feature>
   <feature>appSecurity-2.0</feature>
   <feature>servlet-3.0</feature>
   <feature>usr:MFPDecoderFeature-1.0</feature>
   ```

2. Add the following declarations for Application Center:

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

   <!-- 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">
     </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 6-236.

3. If the database is Oracle, add the **commonLibraryRef** attribute to the class loader of the Application Center 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 Application Center" on page 6-219.

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 LIBERTY_HOME/wlp/usr/servers/server_name
     ```

   - On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\servers\server_name\apps
copy /B product_install_dir\ApplicationCenter\console\applicationcenter.ear
LIBERTY_HOME\wlp\usr\servers\server_name\apps\applicationcenter.ear
```

5. Copy the password decoder user feature.
   - On UNIX and Linux systems:

```
mkdir -p LIBERTY_HOME/wlp/usr/extension/lib/features
cp product_install_dir/features/com.ibm.websphere.crypto_1.0.0.jar LIBERTY_HOME/wlp/usr/extension/lib/
cp product_install_dir/features/MFPDecoderFeature-1.0.mf LIBERTY_HOME/wlp/usr/extension/lib/features/
```

   - On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\extension\lib
copy /B product_install_dir\features\com.ibm.websphere.crypto_1.0.0.jar
LIBERTY_HOME\wlp\usr\extension\lib\com.ibm.websphere.crypto_1.0.0.jar
mkdir LIBERTY_HOME\wlp\usr\extension\lib\features
copy /B product_install_dir\features\MFPDecoderFeature-1.0.mf
LIBERTY_HOME\wlp\usr\extension\lib\features\MFPDecoderFeature-1.0.mf
```

6. 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 6-236.

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 6-207.
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.
8. Configure a JNDI environment entry to indicate the directory with binary files of the `aapt` program, from the Android SDK `platform-tools` package.
   a. Determine a suitable directory for the `aapt` binary files in the WebSphere Application Server installation directory.
      - For a stand-alone server, you can use a directory such as *WAS_INSTALL_DIR*`/optionalLibraries/IBM/mobilefirst/android-sdk`.
      - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*`/profiles/`*profile-name*`/config/cells/`*cell-name*`/mobilefirst/android-sdk`.
      - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*`/profiles/`*profile-name*`/config/cells/`*cell-name*`/clusters/`*cluster-name*`/mobilefirst/android-sdk`.
      - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*`/profiles/`*profile-name*`/config/cells/`*cell-name*`/nodes/`*node-name*`/mobilefirst/android-sdk`.
      - For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*`/profiles/`*profile-name*`/config/cells/`*cell-name*`/nodes/`*node-name*`/servers/`*server-name*`/mobilefirst/android-sdk`.
   b. Copy the *product_install_dir*`/ApplicationCenter/tools/android-sdk` directory to the directory that you determined in Substep a.
   c. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.
   d. Configure the environment entry (JNDI property) `android.aapt.dir` and set as its value the directory that you determined in Substep a. The *WAS_INSTALL_DIR*`/profiles/`*profile-name* profile is replaced with the WebSphere Application Server variable reference `${USER_INSTALL_ROOT}`.

Results

You can now access the Application Center at `http://<server>:<port>/appcenterconsole`, where *server* is the host name of your server and *port* is the port number (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 6-234.

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.

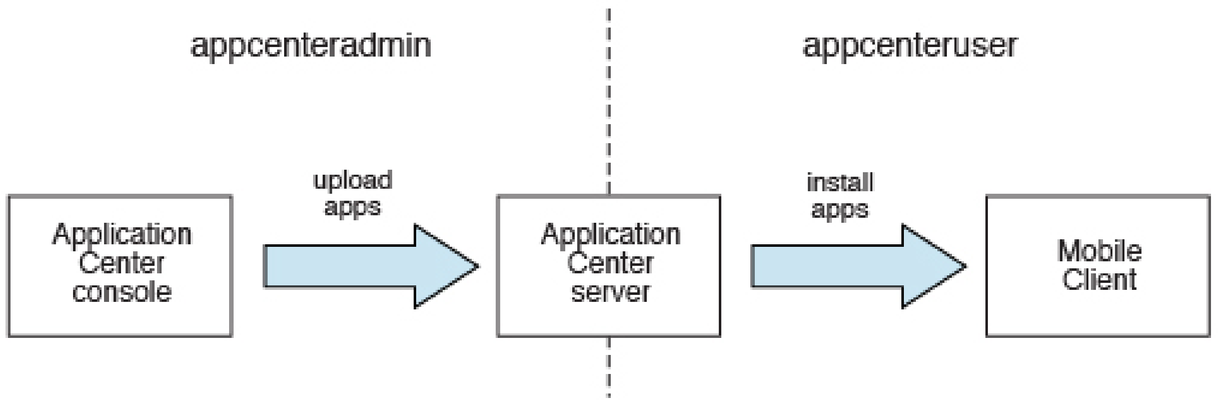


Figure 6-10. Java EE security roles of the Application Center and the components that they influence

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 6-237 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 13-6 for how to build the Application Center mobile client.

Related concepts:

"Managing users with LDAP" on page 6-237
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 either generate the app by using the provided Eclipse and Visual Studio projects or use the version of the client provided for Android, iOS, or Windows 8 Universal, directly.

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

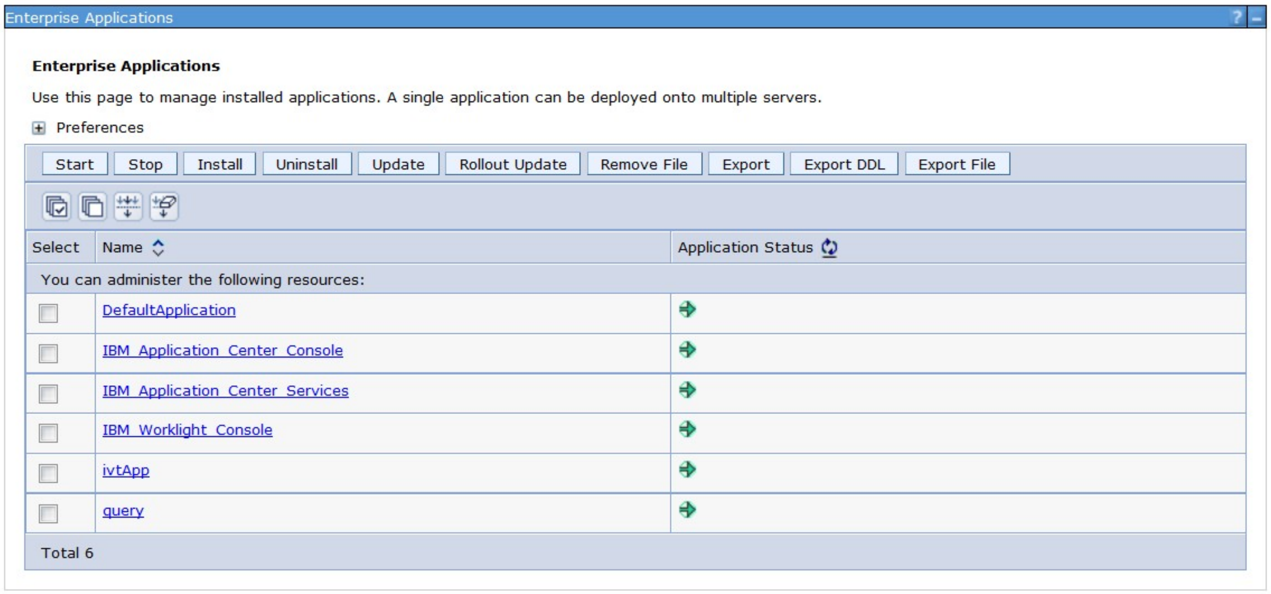
Review the definition of roles at "Configuring user authentication for Application Center" on page 6-233.

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.

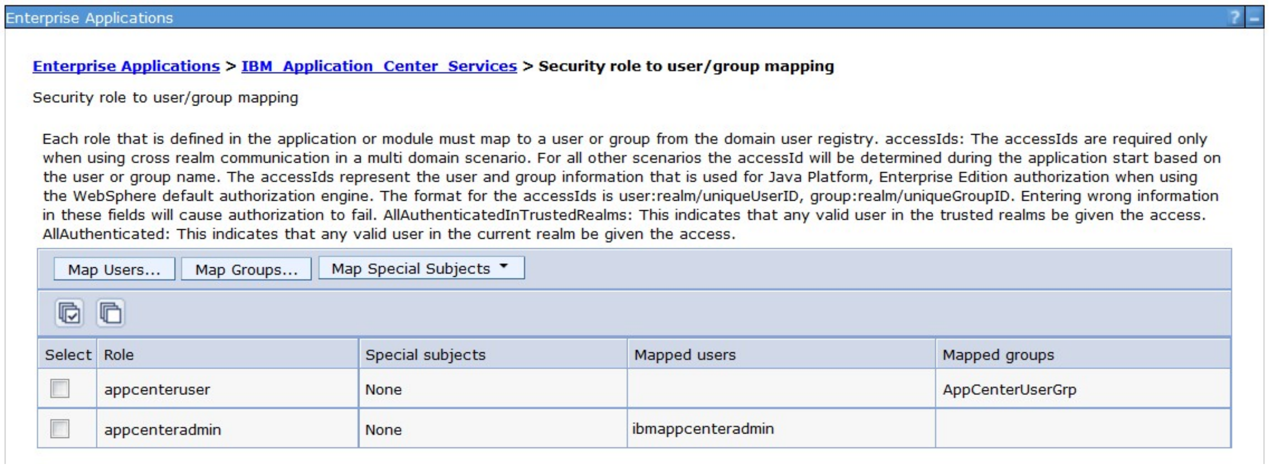
Preferences

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 6-11. 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

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 6-12. Mapping the **appcenteruser** *and* **appcenteradmin** *roles: user groups*

f. Perform the necessary customization.

g. Click **OK**.

h. Repeat steps c to g to map the roles for the console web application; in step d, select **IBM_Application_Center_Console**.

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

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 6-233.

In WebSphere Application Server Liberty profile, you configure the roles of `appcenteruser` and `appcenteradmin` in the `server.xml` configuration file of the server.

About this task

To configure the security roles, you must edit the `server.xml` file. In the `<application-bnd>` element of each `<application>` element, create two `<security-role>` elements. One `<security-role>` element is for the **`appcenteruser`** role and the other is for the **`appcenteradmin`** role. Map the roles to the appropriate user group name **`appcenterusergroup`** or **`appcenteradmingroup`**. These groups are defined through the `<basicRegistry>` element. You can customize this element or replace it entirely with an `<ldapRegistry>` element or a `<safRegistry>` element.

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

Procedure

1. Edit the `server.xml` file.

   For example:

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

   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 6-233.

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 V8.x:

LDAP authentication is 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 for WebSphere Application Server V8.x."

Configuring LDAP ACL management for WebSphere Application Server V8.x:

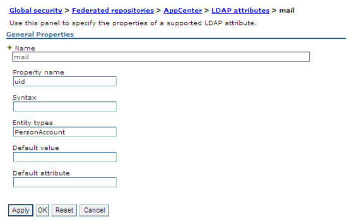
Use LDAP to define the users and groups who can install mobile applications with the Application Center with the Virtual Member Manager (VMM) API.

About this task

To configure ACL with LDAP, you 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 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:

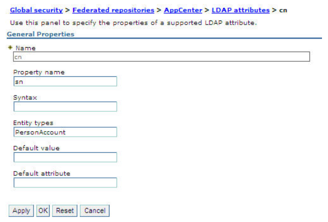
Entity types: PersonAccount

Default value:

Default attribute:

Figure 6-13. Associating LDAP login with uid property (WebSphere Application Server V8.0)

8. Select **Add** > **Supported**.
   a. For **`Name`** enter your LDAP attribute for full user name.
   b. For **`Property name`** enter **`sn`**.
   c. For **`Entity types`** enter the LDAP entity type.
   d. Click **OK**.


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 6-14. Associating LDAP full user name with sn property (WebSphere Application Server V8.0)

9. Select **Add** > **Supported** to configure a group name:
   a. For **`Name`** enter the LDAP attribute for your group name.

b. For **`Property name`** enter **`cn`**.

c. For **`Entity types`** enter the LDAP entity type.

d. Click **OK**.

10. Enable ACL management with LDAP:

a. Select **Servers** > **Server Types** > **WebSphere application servers**.

b. Select the appropriate application server.

In a clustered environment you must configure all the servers in the cluster in the same way.

c. In the **Configuration** tab, under "Server Infrastructure", click the **Java and Process Management** tab and select **Process definition**.

d. In the **Configuration** tab, under Additional Properties, select **Java Virtual Machine**,

e. In the **Configuration** tab, under Additional Properties, select **Custom properties**.

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

Property-value pairs:

- `ibm.appcenter.ldap.vmm.active = true`
- `ibm.appcenter.ldap.active = true`
- `ibm.appcenter.ldap.cache.expiration.seconds =` *`delay_in_seconds`*

g. 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.

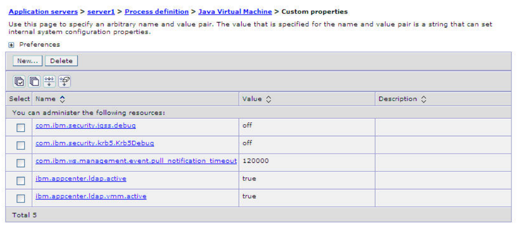
Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **`ibm.appcenter.ldap.cache.expiration.seconds`**. The Application Center maintains a cache of LDAP data and the changes become visible only after the cache expires. By default, 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.

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.

Preferences

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 6-15. ACL management for Application Center with LDAP on WebSphere Application Server V8

What to do next

1. Save the configuration and restart the server.

2. 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 **appcenteruser** or **appcenteradmin**roles.
3. In the *<was_home>*\bin directory, where *<was_home>* is the home directory of your WebSphere Application Server, run the **wsadmin** command.
4.

   After connecting with the WebSphere Application Server administrative user, run the following command:

   ```
   $AdminTask mapIdMgrGroupToRole {-roleName IdMgrReader -groupId your_LDAP_group_id}
   ```
5. Run the same command for all the groups mapped to the **appcenteruser** and **appcenteradmin**roles.

   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}
   ```
6. 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, the following feature must be defined in the `server.xml` file:

```
<feature>jndi-1.0</feature>
```

Add an entry for each property in the `<server>` section of the `server.xml` file. This entry should have the following syntax:

```
<jndiEntry jndiName="JNDI_property_name" value="property_value"/>
```

Where:

`JNDI_property_name` is the name of the property you are adding.

`property_value` is the value of the property you are adding.

Table 6-47. JNDI properties for configuring ACL management with LDAP in the server.xml file

Property	Description
ibm.appcenter.ldap.active	Set to `true` to enable LDAP; set to `false` to disable LDAP.
ibm.appcenter.ldap.federated.active	Since WebSphere Application Server Liberty profile V8.5.5: set to `true` to enable use of the federated registry; set to `false` to disable use of the federated registry, which is the default setting.
ibm.appcenter.ldap.connectionURL	LDAP connection URL.
ibm.appcenter.ldap.user.base	Search base of users.
ibm.appcenter.ldap.user.loginName	LDAP login attribute.
ibm.appcenter.ldap.user.displayName	LDAP attribute for the user name to be displayed, for example, a person's full name.
ibm.appcenter.ldap.group.base	Search base of groups.
ibm.appcenter.ldap.group.name	LDAP attribute for the group name.
ibm.appcenter.ldap.group.uniquemember	LDAP attribute that identifies the members of a group.
ibm.appcenter.ldap.user.groupmembership	LDAP attribute that identifies the groups to which a user belongs.
ibm.appcenter.ldap.group.nesting	Management of nested groups: if nested groups are not managed, set the value to `false`.
ibm.appcenter.ldap.user.filter	LDAP user search filter for the attribute of user login name. Use `%v` as the placeholder for the login name attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.displayName.filter	LDAP user search filter for the attribute of user display name. Use `%v` as the placeholder for the display name attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.group.filter	LDAP group search filter. Use `%v` as the placeholder for the group attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.

Table 6-47. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
`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 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 generated by the tool. The supported encoding types are `xor` and `aes`. Edit the Liberty profile `server.xml` file to check whether the *classloader* is enabled to load the JAR file that decodes the password.

`ibm.appcenter.ldap.cache.expiration.seconds`	Delay in seconds before the LDAP cache expires. If no value is entered, the default value is 86400, which is equal to 24 hours.
	
	Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **`ibm.appcenter.ldap.cache.expiration.seconds`**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data:
	`acdeploytool.sh -clearLdapCache -s serverurl`
	`-c context -u user -p password`
	
	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 “JNDI properties for Application Center” on page 6-261 for a complete list of LAPD properties that you can set.

Example of setting properties for ACL management with LDAP

This example shows the settings of the properties in the `server.xml` file required for ACL management with LDAP.

```
<jndiEntry jndiName="ibm.appcenter.ldap.active" value="true"/>
<jndiEntry jndiName="ibm.appcenter.ldap.connectionURL" value="ldap://employees.com:636"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.loginName" value="uid"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.base" value="dc=ibm,dc=com"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.base" value="dc=ibm,dc=com"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName" value="sn"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.name" value="cn"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.uniquemember" value="uniqueMember"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.groupmembership" value=ibm-allGroups"/>
<jndiEntry jndiName="ibm.appcenter.ldap.cache.expiration.seconds" value=43200"/>
<jndiEntry jndiName="ibm.appcenter.ldap.security.sasl" value='"EXTERNAL"'/>
<jndiEntry jndiName="ibm.appcenter.ldap.referral" value='"follow"'/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.filter" value='"(&amp;(uid=%v)(objectclass=inetOrgPerson))"'/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName.filter" value='"(&amp;(cn=%v)(objectclass=inetOrgPerson))"'/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.filter" value='"(&amp;(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))"'/>
```

LDAP with Apache Tomcat:

Configure the Apache Tomcat application 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 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 should have the following syntax:

```
<Environment name="JNDI_property_name" value="property_value" type="java.lang.String" 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.

Table 6-48. 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 6-48. 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.

`ibm.appcenter.ldap.cache.expiration.seconds`	Delay in seconds before the LDAP cache expires. If no value is entered, the default value is 86400, which is equal to 24 hours.
	
	Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **`ibm.appcenter.ldap.cache.expiration.seconds`**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data:
	`acdeploytool.sh -clearLdapCache -s serverurl`
	`c context -u user -p password`
	
	See Using the stand-alone tool to clear the LDAP cache for details.

Table 6-48. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat (continued)

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

See "JNDI properties for Application Center" on page 6-261 for a complete list of LAPD properties that you can set.

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

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

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`**).

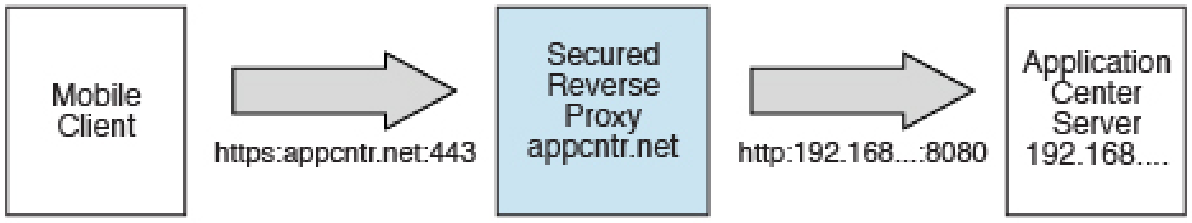

Mobile Client
https:appcntr.net:443
Secured Reverse Proxy appcntr.net
http:192.168...:8080
Application Center Server 192.168....

Figure 6-16. Configuration with secured reverse proxy

Table 6-49. 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 “JNDI properties for Application Center” on page 6-261 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 "JNDI properties for Application Center" on page 6-261.

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 6-50. 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.

Table 6-50. Properties in the server.xml file for configuring the endpoint of the application resources (continued)

Property	Description
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 "JNDI properties for Application Center" on page 6-261.

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 6-51. 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.
`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 "JNDI properties for Application Center" on page 6-261.

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.

You can configure the Application Center with SSL or without SSL, **unless** you intend to install applications on iOS devices. For iOS applications, you must configure the Application Center server with SSL.

SSL transmits data over the network in a secured channel. You must purchase an official SSL certificate from an SSL certificate authority. The SSL certificate must be compatible with Android and iOS. 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 that is filed in the SSL certificate, the client accepts the connection. For the verification to be successful, the client must know the root certificate of the certificate authority. Many root certificates are preinstalled on Android and iOS devices. The exact list of pre-installed root certificates varies between versions of mobile operating systems.

For information about the mobile operating system versions that support its certificates, consult the SSL certificate authority.

If the SSL certificate verification fails, a normal web browser requests confirmation to contact an untrusted site. The same behavior occurs when you use a self-signed certificate that was not purchased from a certificate authority. When mobile applications are installed, this control is not performed by a normal web browser, but by operating system calls.

Some versions of Android, iOS, and Windows Phone operating systems do not support this confirmation dialog in system calls. This limitation is a reason to avoid self-signed certificates or SSL certificates that are not suited to mobile operating systems. On Android, iOS, and Windows Phone operating systems, you can install a self-signed CA certificate on the device to enable the device to handle system calls regarding this self-signed certificate. This practice is not appropriate for Application Center in a production environment, but it can be suitable during the testing period. For details, see "Managing and installing self-signed CA certificates in an Application Center test environment" on page 6-260.

Configuring SSL for WebSphere Application Server full profile:

Request a Secure Sockets Layer (SSL) certificate and process the received documents to import them into the keystore.

About this task

This procedure indicates how to request an SSL certificate and import it and the chain certificate into your keystore.

Procedure

1. Create a request to a certificate authority; in the WebSphere administrative console, select **Security** > **SSL certificate and key management** > **Key stores and certificates** > ***keystore*** > **Personal certificate requests** > **New**.

   Where *keystore* identifies your keystore.

   The request is sent to the certificate authority.
2. When you receive the SSL certificate, import it and the corresponding chain certificate into your keystore by following the instructions provided by the certificate authority. In the WebSphere administrative console, you can find the corresponding option in **Security** > **SSL certificate and key management** > **Manage endpoint security configurations** > *node SSL settings* > **Key stores and certificates** > ***keystore*** > **Personal certificates** > *certificate* > **Receive a certificate from a certificate authority**.

   Where:
   - *node SSL settings* shows the SSL settings of the nodes in your configuration.
   - *keystore* identifies your keystore.
   - *certificate* identifies the certificate that you received.

3. Create an SSL configuration. See the instructions in the user documentation that corresponds to the version of the WebSphere Application Server full profile that supports your applications.

   You can find configuration details in the WebSphere administrative console at **Security** > **SSL certificate and key management** > **Manage endpoint security configurations** > **SSL Configurations**.

Configuring SSL for Liberty profile:

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

About this task

Follow the steps in this procedure to configure SSL on Liberty profile.

Procedure

1. Create a keystore for your web server; use the **`securityUtility`** with the `createSSLCertificate` option. See Enabling SSL communication for the Liberty profile for more information.
2. Import the SSL certificate and the corresponding chain certificate into your keystore by following the instructions provided by the certificate authority.
3. Enable the ssl-1.0 Liberty feature in the `server.xml` file.
   ```
   <featureManager>
     <feature>ssl-1.0</feature>
   </featureManager>
   ```
4. Add the keystore service object entry to the `server.xml` file. The **`keyStore`** element is called **`defaultKeyStore`** and contains the keystore password. For example:
   ```
   <keyStore id="defaultKeyStore" location="/path/to/myKeyStore.p12"
             password="myPassword" type="PKCS12"/>
   ```
5. Make sure that the value of the **`httpEndpoint`** element in the `server.xml` file defines the **`httpsPort`** attribute. For example:
   ```
   <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="9443" >
   ```
6. Restart the web server. Now you can access the web server by `https://`*`myserver`*`:9443/...`

Configuring SSL for Apache Tomcat:

Create a keystore, import the Secure Socket Layer (SSL) certificate, and edit the `conf/server.xml` file to define a connector for SSL on Apache Tomcat.

About this task

Follow the steps in this procedure to configure SSL on Apache Tomcat. See SSL Configuration HOW-TO for more details and examples of configuring SSL for Apache Tomcat.

Procedure

1. Create a keystore for your web server. You can use the Java **`keytool`** command to create a keystore.
   ```
   keytool -genkey -alias tomcat -keyalg RSA -keystore /path/to/keystore.jks
   ```
2. Import the SSL certificate and the corresponding chain certificate into your keystore by following the instructions provided by the certificate authority.

3. Edit the `conf/server.xml` file to define a connector to use SSL. This connector must point to your keystore.

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS"
           keystoreFile="/path/to/keystore.jks"
           keystorePass="mypassword" />
```

4. Restart the web server. Now you can access the web server by `https://`*`myserver`*`:8443/...`

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.

As an administrator, you can easily distribute self-signed CA certificates to devices.

The following procedure focuses mostly on the iOS and Android environments. Support for X.509 certificates comes from the individual mobile platforms, not from IBM MobileFirst Platform Foundation. For more information about specific requirements for X.509 certificates, see the documentation of each mobile platform.

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**.

JNDI properties for Application Center

You can configure some JNDI properties for Application Center.

Table 6-52. List of the JNDI properties for Application Center.

Property	Description
`appcenter.database.type`	The database type, which is required only 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 needed only when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.openjpa.ConnectionPassword`	The password for the database connection. Set this property only when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.openjpa.ConnectionURL`	The URL for the database connection driver class. Set this property only when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.openjpa.ConnectionUserName`	The user name or the database connection. Set this property only when the database is not specified in **`appcenter.jndi.name`**.
`ibm.appcenter.apns.p12.certificate.` **`isDevelopmentCertificate`**	Set this property to `true` to specify whether the certificate that enables Application Center to send push notifications about updates of iOS applications is a development certificate. Set the property to `false` if it is not a development certificate. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 13-16.
`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 13-16.

Table 6-52. List of the JNDI properties for Application Center (continued).

Property	Description
ibm.appcenter.apns.p12.certificate.password	The password of the certificate that enables Application Center to send push notifications about updates of iOS applications is a development certificate. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 13-16.
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 reset this parameter to `false`.
ibm.appcenter.gcm.signature.googleapikey	The Google API key that enables Application Center to send push notifications about updates for Android applications. For example, `AIxaScCHg0VSGdgfOZKtzDJ44-oi0muUasMZvAs`. See "Configuring the Application Center server for connection to Google Cloud Messaging" on page 13-14.
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, because one-time URLs are generated with a cryptographic mechanism so that nobody can guess the URL but do not require the user to log in. Setting this property to `false` provides maximal security, because 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 13-54.
ibm.appcenter.ldap.active	Specifies whether Application Center is configured for LDAP. Set this property to `true` to enable LDAP or to `false` to disable LDAP. See "Managing users with LDAP" on page 6-237.
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 number of seconds during which an entry in the LDAP cache is valid. Set this property to a value greater 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 clear the cache of LDAP data manually, enter this command: `acdeploytool.sh -clearLdapCache -s serverurl -c context -u user -p password` See Using the stand-alone tool to clear the LDAP cache.
ibm.appcenter.ldap.connectionURL	The URL to access the LDAP server when no Virtual Member Manager (VMM) is used. See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.

Table 6-52. List of the JNDI properties for Application Center (continued).

Property	**Description**
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 6-237.
ibm.appcenter.ldap.group.base	The search base to find groups when you use LDAP without Virtual Member Manager (VMM). See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
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 Virtual Member Manager (VMM). See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
ibm.appcenter.ldap.group.nesting	Specifies whether the LDAP contains nested groups (that is, groups in groups) when you use LDAP without Virtual Member Manager (VMM). Setting this property to `false` speeds up LDAP access because the groups are not searched recursively. See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
ibm.appcenter.ldap.group.uniquemember	Specifies the members of a group when you use LDAP without Virtual Member Manager (VMM). This property is the inverse of **ibm.appcenter.ldap.user.groupmembership**. See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
ibm.appcenter.ldap.referral	Specifies whether referrals are supported by the JNDI API. If no value is specified, 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.

Table 6-52. List of the JNDI properties for Application Center (continued).

Property	Description
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. 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. 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 6-249.
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`. When this property is set, security authentication is required to connect to LDAP without Virtual Member Manager (VMM). See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
ibm.appcenter.ldap.user.base	The search base to find users when you use LDAP without Virtual Member Manager (VMM). See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
ibm.appcenter.ldap.user.displayName	The display name attribute, such as the user's real name, when you use LDAP without Virtual Member Manager (VMM). See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
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 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.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 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.user.groupmembership	Specifies the groups of a member when you use LDAP without Virtual Member Manager (VMM). This property is the inverse of **ibm.appcenter.ldap.group.uniquemember**. This property is optional, but if it is specified, LDAP access is faster. See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.

Table 6-52. List of the JNDI properties for Application Center (continued).

Property	Description
ibm.appcenter.ldap.user.loginName	The login name attribute when you use LDAP without Virtual Member Manager (VMM). See "Configuring LDAP ACL management (Liberty profile)" on page 6-243 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-249.
ibm.appcenter.ldap.vmm.active	Set this property to `true` to specify that LDAP is done through Virtual Member Manager (VMM), or to `false` otherwise. See "Configuring LDAP ACL management for WebSphere Application Server V8.x" on page 6-239.
ibm.appcenter.ldap.vmm.adminpwd	The password when LDAP is done through Virtual Member Manager (VMM). See "Configuring LDAP ACL management for WebSphere Application Server V8.x" on page 6-239.
ibm.appcenter.ldap.vmm.adminuser	The user when LDAP is done through Virtual Member Manager (VMM). See "Configuring LDAP ACL management for WebSphere Application Server V8.x" on page 6-239.
ibm.appcenter.logging.formatjson	This property has an effect only when **ibm.appcenter.logging.tosystemerror** is set to `true`. If this property is enabled, it formats JSON responses in logging messages that are directed to System.Error. Setting this property is helpful when you debug the server.
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 6-252.
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 needed only if the protocol of the external URI and the protocol of the internal URI are different. See "Defining the endpoint of the application resources" on page 6-252.
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 needed only if the protocol of the external and of the internal URI are different. See "Defining the endpoint of the application resources" on page 6-252.
ibm.appcenter.proxy.scheme	This property is just an alternative name for **ibm.appcenter.proxy.protocol**.

Table 6-52. List of the JNDI properties for Application Center (continued).

Property	Description
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 6-252.
ibm.appcenter.services.iconCacheMaxAge	Specifies the number of seconds during which 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.
mfp.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.
mfp.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.

Configuring WebSphere Application Server to support applications in public app stores

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

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

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

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

To use application links to Google play, see "Configuring WebSphere Application Server to support applications in Google play" on page 6-267.

To use application links to Apple iTunes, see "Configuring WebSphere Application Server to support applications in Apple iTunes" on page 6-267.

Configuring WebSphere Application Server to support applications in Google play:

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

About this task

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

Procedure

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

Configuring WebSphere Application Server to support applications in Apple iTunes:

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

About this task

Follow this procedure to import the root certificate of Apple iTunes into the WebSphere 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.ibm.websphere.ssl.protocol.SSLSocketFactory
        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(RESTMBeanServerConnection.java:142)
        at com.ibm.ws.jmx.connector.client.rest.internal.RESTMBeanServerConnection.<init>(RESTMBeanServerConnection.java:114)
        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)
```

Installation reference

Reference information about Ant tasks and configuration sample files for the installation of IBM MobileFirst Platform Server, IBM MobileFirst Platform Application Center, and IBM MobileFirst Analytics.

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

The **`configuredatabase`** Ant task creates the relational databases that are used by MobileFirst Server administration service, MobileFirst Server live update service, MobileFirst Server push service, MobileFirst runtime, and the Application Center services. This Ant task configures a relational database 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, migrates them to the current version.
- If the tables exist for the current version of IBM MobileFirst Platform Foundation, 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 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.

Attributes and elements for `configuredatabase` task

The **configuredatabase** task has the following attributes:

Table 6-53. Attributes for the **configuredatabase** *Ant task*

Attribute	Description	Required	Default
kind	The type of database: In MobileFirst Server: `MobileFirstRuntime`, `MobileFirstConfig`, `MobileFirstAdmin`, or `push`. In Application Center: `ApplicationCenter`.	Yes	None
includeConfigurationTables	To specify whether to perform database operations on both the live update service and the administration service or on the administration service only. The value is either `true` or `false`.	No	`true`
execute	To specify whether to execute the **configuredatabase** Ant task. The value is either `true` or `false`.	No	`true`

kind IBM MobileFirst Platform Foundation V8.0.0 supports four kinds of database: MobileFirst runtime uses `MobileFirstRuntime` database. MobileFirst Server administration service uses the `MobileFirstAdmin` database. MobileFirst Server live update service uses the `MobileFirstConfig` database. By default, it is created with `MobileFirstAdmin` kind. MobileFirst Server push service uses the `push` database.Application Center uses the **ApplicationCenter** database.

includeConfigurationTables

The **includeConfigurationTables** attribute can be used only when **kind** attribute is `MobileFirstAdmin`. The valid value can be `true` or `false`. When this attribute is set to `true`, the **configuredatabase** task performs database operations on both the administration service database and the live update service database in a single run. When this attribute is set to `false`, the **configuredatabase** task performs database operations only on the administration service database.

execute

The **execute** attribute enables or disables the execution of the **configuredatabase** Ant task. The valid value can be `true` or `false`. When this attribute is set to `false`, the **configuredatabase** task performs no configuration or database operations.

The **configuredatabase** task supports the following elements:

Table 6-54. Inner elements for the **configuredatabase** *Ant task*

Element	Description	Count
`<derby>`	The parameters for Derby.	0..1
`<db2>`	The parameters for DB2.	0..1
`<mysql>`	The parameters for MySQL.	0..1

Table 6-54. Inner elements for the **configuredatabase** Ant task (continued)

Element	Description	Count
`<oracle>`	The parameters for Oracle.	0..1
`<driverclasspath>`	The JDBC driver class path.	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 6-55. Attributes for the `<property>` element

Attribute	Description	Required	Default
name	The name of the property.	Yes	None
value	The value for the property.	Yes	None

Apache Derby

The `<derby>` element has the following attributes:

Table 6-56. Attributes for the `<derby>` element

Attribute	Description	Required	Default
database	The database name.	No	`MFPDATA`, `MFPADM`, `MFPCFG`, `MFPPUSH`, or `APPCNTR`, depending on kind.
datadir	The directory that contains the databases.	Yes	None
schema	The schema name.	No	`MFPDATA`, `MFPCFG`, `MFPADMINISTRATOR`, `MFPPUSH`, or `APPCENTER`, depending on kind.

The `<derby>` element supports the following element:

Table 6-57. Inner element for the `<derby>` element

Element	Description	Count
`<property>`	The 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 6-58. Attributes for the `<db2>` element

Attribute	Description	Required	Default
database	The database name.	No	`MFPDATA`, `MFPADM`, `MFPCFG`, `MFPPUSH`, or `APPCNTR`, depending on kind.

Table 6-58. Attributes for the <db2> element (continued)

Attribute	Description	Required	Default
server	The host name of the database server.	Yes	None
port	The port on the database server.	No	50000
user	The user name for accessing databases.	Yes	None
password	The password for accessing databases.	No	Queried interactively
instance	The name of the DB2 instance.	No	Depends on the server
schema	The 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 6-59. Inner elements for the <db2> element

Element	Description	Count
<property>	The JDBC connection property.	0..∞
<dba>	The 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 the credentials for the database administrators. This element has the following attributes:

Table 6-60. Attributes for the <dba> element for DB2 databases

Attribute	Description	Required	Default
user	The user name for accessing database.	Yes	None
password	The 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 the 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 "DB2 database and user requirements" on page 6-65.

MySQL

The element `<mysql>` has the following attributes:

Table 6-61. Attributes for the `<mysql>` element

Attribute	Description	Required	Default
database	The database name.	No	MFPDATA, MFPADM, MFPCFG, MFPPUSH, or APPCNTR, depending on kind.
server	The host name of the database server.	Yes	None
port	The port on the database server.	No	3306
user	The user name for accessing databases.	Yes	None
password	The 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 6-62. Inner elements for the `<mysql>` element

Element	Description	Count
`<property>`	The JDBC connection property.	0..∞
`<dba>`	The 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 the database administrator credentials. This element has the following attributes:

Table 6-63. Attributes for the `<dba>` element for MySQL databases

Attribute	Description	Required	Default
user	The user name for accessing databases.	Yes	None
password	The 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 6-64. Attribute for the `<client>` element for MySQL databases

Attribute	Description	Required	Default
hostname	The 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 6-65. Alternative attributes for the `<mysql>` element

Attribute	Description	Required	Default
url	The database connection URL.	Yes	None
user	The user name for accessing databases.	Yes	None
password	The 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 6-66. Attributes for the `<oracle>` element

Attribute	Description	Required	Default
database	The database name, or Oracle service name. **Note:** You must always use a service name to connect to a PDB database.	No	`ORCL`
server	The host name of the database server.	Yes	None
port	The port on the database server.	No	1521
user	The user name for accessing databases. See the note under this table.	Yes	None
password	The password for accessing databases.	No	Queried interactively
sysPassword	The password for the user `SYS`.	No	Queried interactively if the database does not yet exist
systemPassword	The 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 6-67. Inner elements for the `<oracle>` element

Element	Description	Count
`<property>`	The JDBC connection property.	0..∞
`<dba>`	The database administrator credentials.	0..1

For information about the available connection properties, see Class OracleDriver.

The inner element `<dba>` specifies the database administrator credentials. This element has the following attributes:

Table 6-68. Attributes for the `<dba>` element for Oracle databases

Attribute	Description	Required	Default
user	The user name for accessing databases.	Yes	None
password	The 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 you run the Ant task. To control other details, you must use the manual instructions in section “Oracle database and user requirements” on page 6-65.

Alternatively, you can use the `<oracle>` element with the following attributes:

Table 6-69. Alternative attributes for the `<oracle>` element

Attribute	Description	Required	Default
url	The database connection URL.	Yes	None
user	The user name for accessing databases.	Yes	None
password	The 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>`.

Ant tasks for installation of MobileFirst Operations Console, MobileFirst Server artifacts, MobileFirst Server administration, and live update services

The **installmobilefirstadmin**, **updatemobilefirstadmin**, and **uninstallmobilefirstadmin** Ant tasks are provided for the installation of MobileFirst Operations Console, the artifacts component, the administration service, and the live update service.

Task effects

installmobilefirstadmin

The **installmobilefirstadmin** Ant task configures an application server to run the WAR files of the administration and live update services as web applications, and optionally, to install the MobileFirst Operations Console. This task has the following effects:

- It declares the administration service web application in the specified context root, by default `/mfpadmin`.
- It declares the live update service web application in a context root derived from the specified context root of the administration service. By default, `/mfpadminconfig`.
- For the relational databases, it declares data sources and on WebSphere Application Server full profile, JDBC providers for the administration services.
- It deploys the administration service and the live update service on the application server.
- Optionally, it declaresMobileFirst Operations Console as a web application in the specified context root, by default `/mfpconsole`. 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">
  <installmobilefirstadmin servicewar="${mfp.service.war.file}">
    <console install="${mfp.admin.console.install}" warFile="${mfp.console.war.file}"/>
```

- Optionally, it declares the MobileFirst Server artifacts web application in the specified context root `/mfp-dev-artifacts` when MobileFirst Operations Console is installed.
- It configures the configuration properties for the administration service 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 MobileFirst Operations Console, and the administration and live update services web applications.
- It configures the application server for use of JMX.
- Optionally, it configures the communication with the MobileFirst Server push service.
- Optionally, it sets the MobileFirst JNDI environment entries to configure the application server as a server farm member for the MobileFirst Server administration part.

updatemobilefirstadmin

The **updatemobilefirstadmin** Ant task updates an already-configured MobileFirst Server web application on an application server. This task has the following effects:

- It updates the administration service WAR file. This file must have the same base name as the corresponding WAR file that was previously deployed.
- It updates the live update service 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.

uninstallmobilefirstadmin

The **uninstallmobilefirstadmin** Ant task undoes the effects of an earlier run of **installmobilefirstadmin**. This task has the following effects:

- It removes the configuration of the administration service 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 WAR files of the administration and live update services, and MobileFirst Operations Console 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 the administration and live update services.
- For the relational DBMS, it removes the database drivers that were used by the administration and live update services from the application server.
- It removes the associated JNDI environment entries.
- On WebSphere Application Server Liberty and Apache Tomcat, it removes the users configured by the **installmobilefirstadmin** invocation.
- It removes the JMX configuration.

Attributes and elements

The **installmobilefirstadmin**, **updatemobilefirstadmin**, and **uninstallmobilefirstadmin** Ant tasks have the following attributes:

Table 6-70. Attributes for the **installmobilefirstadmin**, **updatemobilefirstadmin**, *and* **uninstallmobilefirstadmin** *Ant tasks*

Attribute	Description	Required	Default
contextroot	The common prefix for URLs to the administration service to get information about MobileFirst runtime environments, applications, and adapters.	No	`/mfpadmin`
id	To distinguish different deployments.	No	Empty
environmentId	To distinguish different MobileFirst environments.	No	Empty
servicewar	The WAR file for the administration service.	No	The `mfp-admin-service.war` file is in the same directory as the `mfp-ant-deployer.jar` file.

Table 6-70. Attributes for the **installmobilefirstadmin**, **updatemobilefirstadmin**, *and* **uninstallmobilefirstadmin** *Ant tasks (continued)*

Attribute	Description	Required	Default
shortcutsDir	The directory where to place shortcuts.	No	None
wasStartingWeight	The 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 the administration service.

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 service 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 service 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 service 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.
- `mobilefirst-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.
- `mobilefirst-admin-service.sh` - this file is a UNIX shell script that provides the same output as the `mobilefirst-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 service 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 service database and cannot handle server requests.

The **installmobilefirstadmin**, **updatemobilefirstadmin**, and **uninstallmobilefirstadmin** Ant tasks support the following elements:

Table 6-71. Inner elements for the **installmobilefirstadmin**, **updatemobilefirstadmin**, *and* **uninstallmobilefirstadmin** *Ant tasks*

Element	Description	Count
`<applicationserver>`	The application server.	1
`<configuration>`	The live update service.	1
`<console>`	The administration console.	0..1
`<database>`	The databases.	1
`<jmx>`	To enable Java Management Extensions.	1
`<property>`	The properties.	0..∞
`<push>`	The push service.	0..1
`<user>`	The 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 6-72. Attributes of the `<console>` element

Attribute	Description	Required	Default
contextroot	The URI of the MobileFirst Operations Console.	No	`/mfpconsole`
install	To indicate whether the MobileFirst Operations Console must be installed.	No	Yes
warfile	The console WAR file.	No	The `mfp-admin-ui.war` file is in the same directory as the`mfp-ant-deployer.jar` file.

The `<console>` element supports the following element:

Table 6-73. Inner element for the `<console>` element

Element	Description	Count
`<artifacts>`	The MobileFirst Server artifacts.	0..1

Table 6-73. Inner element for the `<console>` element (continued)

Element	Description	Count
`<property>`	The properties.	0..∞

The `<artifacts>` element has the following attributes:

Table 6-74. Attributes for the `<artifacts>` element

Attribute	Description	Required	Default value
install	To indicate whether the artifacts component must be installed.	No	`true`
warFile	The artifacts WAR file.	No	The `mfp-dev-artifacts.war` file is in the same directory as the `mfp-ant-deployer.jar` file

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 service and the MobileFirst Operations Console WAR files.

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

Table 6-75. Attributes for the `<property>` element

Attribute	Description	Required	Default value
name	The name of the property.	Yes	None
value	The 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 service and the MobileFirst Operations Console WAR files.

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

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.

Table 6-76. Inner elements of the `<applicationserver>` element

Element	Description	Count
`<websphereapplicationserver>` or `<was>`	The parameters for WebSphere Application Server. The `<websphereapplicationserver>` element (or `<was>` in its short form) denotes a WebSphere Application Server instance. WebSphere Application Server full profile (Base, and Network Deployment) are supported, so is WebSphere Application Server Liberty Core and WebSphere Application Server Liberty Network Deployment.	0..1
`<tomcat>`	The parameters for Apache Tomcat.	0..1

The attributes and inner elements of these elements are described in Table 6-105 on page 6-295 through Table 6-114 on page 6-298 of "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

However, for the inner element of the `<was>` element for Liberty collective, see the following table:

Table 6-77. Inner element of the `<was>` element for Liberty collective

Element	Description	Count
`<collectiveController>`	A Liberty collective controller.	0..1

The `<collectiveController>` element has the following attributes:

Table 6-78. Attributes of the `<collectiveController>` element

Attribute	Description	Required	Default value
serverName	The name of the collective controller.	Yes	None
controllerAdminName	The administrative user name that is defined in the collective controller. This is the same user that is used to join new members to the collective.	Yes	None
controllerAdminPassword	The administrative user password.	Yes	None
createControllerAdmin	To indicate whether the administrative user must be created in the basic registry of the collective controller. Possible values are `true` or `false`.	No	`true`

To specify the live update service configuration

Use the `<configuration>` element to define the parameters that depend on the live update service. The `<configuration>` element has the following attributes.

Table 6-79. Attributes of the `<configuration>` element

Attribute	Description	Required	Default value
`install`	To indicate whether the live update service must be installed.	Yes	`true`
`configAdminUser`	The administrator for the live update service.	No. However, it is required for a server farm topology.	If not defined, a user is generated. In a server farm topology, the user name must be the same for all the members of the farm.
`configAdminPassword`	The administrator password for live update service user.	If a user is specified for **`configAdminUser`**.	None. In a server farm topology, the password must be the same for all the members of the farm.
`createConfigAdminUser`	To indicate whether to create an admin user in the basic registry of the application server, if it is missing.	No	`true`
`warFile`	The live update service WAR file.	No	The `mfp-live-update.war` file is in the same directory as the `mfp-ant-deployer.jar` file.

The `<configuration>` element supports the following elements:

Table 6-80. Inner elements of the `<configuration>` element

Element	Description	Count
`<user>`	The user for the live update service.	0..1
`<property>`	The properties.	0..∞

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

Table 6-81. Attributes of the `<user>` element

Attribute	Description	Required	Default value
`role`	A valid security role for the application. Possible value: `configadmin`.	Yes	None
`name`	The user name.	Yes	None

Table 6-81. Attributes of the `<user>` element (continued)

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

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

- `configadmin`

For more information about which authorizations are implied by the specific roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

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

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

Table 6-82. Attributes for the `<property>` element

Attribute	Description	Required	Default value
name	The name of the property.	Yes	None
value	The 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 service and the MobileFirst Operations Console WAR files.

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration service" on page 6-174.

To specify the push service configuration

Use the `<push>` element to define the parameters to configure the connection to the push service. The `<push>` element has the following attribute.

Table 6-83. Attribute of the `<push>` element

Attribute	Description	Required	Default value
configure	To indicate whether to configure the connection to the push service.	No	`false`

The `<push>` element supports the following element:

Table 6-84. Inner element of the `<push>` element

Element	Description	Count
`<authorization>`	The configuration of the authorization server for the communication with the push service. Mandatory if the **configure** attribute is set to `true`.	1

The `<authorization>` element collects information to configure the authorization server for the authentication communication with other MobileFirst Server components. This element has the following attributes:

Table 6-85. Attributes of the `<authorization>` element

Attribute	Description	Required	Default value
auto	To indicate whether the authorization server URL is computed. The possible values are `true` or `false`.	Required on a WebSphere Application Server Network Deployment cluster or node.	`true`
authorizationURL	The URL of the authorization server.	If mode is not auto.	The context root of the runtime on the local server.
runtimeContextRoot	The context root of the runtime.	No	`/mfp`
adminClientID	The administration service confidential ID in the authorization server.	Yes	None
adminClientSecret	The administration service confidential client password in the authorization server.	Yes	None
pushURL	The URL of the push service.	If mode is not auto.	`/imfpush` on the local server.
pushClientID	The push service confidential client ID in the authorization server.	Yes	None
pushClientSecret	The push service confidential password in the authorization server.	Yes	None

auto
If the value is set to true, the URL of the authorization server and the push service is computed automatically by using the context root of the runtime on the local application server. The auto mode is not supported if you deploy on WebSphere Application Server Network Deployment on a cluster.

authorizationURL
The URL of the authorization server. If the authorization server is the MobileFirst runtime, the URL is the URL of the runtime. For example, `http://myHost:9080/mfp`.

runtimeContextRoot
The context root of the runtime that is used to compute the URL of the authorization server in the automatic mode.

adminClientID
The ID of this administration service instance as a confidential client of the authorization server.

adminClientSecret
: The secret key of this administration service instance as a confidential client of the authorization server.

pushClientID
: The ID of this push service instance as a confidential client of the authorization server. If provided, the administration service registers it as a confidential client.

pushClientSecret
: The secret key of this push service instance as a confidential client of the authorization server. If provided, the administration service registers it as a confidential client.

To specify JMX communication between the MobileFirst Server administration service and the MobileFirst projects

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

Table 6-86. 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
tomcatSetEnvConfig	To prevent 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

administration service 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 **installmobilefirstadmin** and **uninstallmobilefirstadmin** Ant tasks from adding or removing contents to the setenv.sh or setenv.bat script, in the *Tomcat_Root_Install_Dir*/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 **installmobilefirstadmin** and **uninstallmobilefirstadmin** 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 service 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 **installmobilefirstadmin** and **uninstallmobilefirstadmin** 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 service database

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

You must declare a single database: `<database kind="MobileFirstAdmin">`. 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 6-87. Attributes of the `<database>` element

Attribute	Description	Required	Default
`kind`	The kind of database (`MobileFirstAdmin`).	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 6-118 on page 6-300 through Table 6-128 on page 6-304 in "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

Table 6-88. 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 (relational DBMS only).	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 6-89. 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.

- `mfpmonitor`
- `mfpoperator`
- `mfpdeployer`

- `mfpadmin`

For information about which authorizations are implied by the specific roles, see the chapter about the "REST API for the MobileFirst Server administration service" on page 8-7.

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 Server push service

The **`installmobilefirstpush`**, **`updatemobilefirstpush`**, and **`uninstallmobilefirstpush`** Ant tasks are provided for the installation of the push service.

Task effects

`installmobilefirstpush`

The **`installmobilefirstpush`** Ant task configures an application server to run the push service WAR file as web application. This task has the following effects:

- It declares the push service web application in the `/imfpush` context root. The context root cannot be changed.
- For the relational databases, it declares data sources and, on WebSphere Application Server Full Profile, JDBC providers for push service.
- It configures the configuration properties for the push service by using JNDI environment entries. These JNDI environment entries configure the OAuth communication with the MobileFirst authorization server, MobileFirst Analytics, and with Cloudant in case Cloudant is used.

`updatemobilefirstpush`

The **`updatemobilefirstpush`** Ant task updates an already-configured MobileFirst Server web application on an application server. This task updates the push service WAR file. This file must have the same base name as the corresponding WAR file that was previously deployed.

`uninstallmobilefirstpush`

The **`uninstallmobilefirstpush`** Ant task undoes the effects of an earlier run of **`installmobilefirstpush`**. This task has the following effects:

- It removes the configuration of the push service 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 push service 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 the push service.
- It removes the associated JNDI environment entries.

Attributes and elements

The **`installmobilefirstpush`**, **`updatemobilefirstpush`**, and **`uninstallmobilefirstpush`** Ant tasks have the following attributes:

Table 6-90. Attributes for the **installmobilefirstpush**, **updatemobilefirstpush**, *and* **uninstallmobilefirstpush** *Ant tasks*

Attribute	Description	Required	Default
id	To distinguish different deployments.	No	Empty
warFile	The WAR file for the push service.	No	The `../PushService/mfp-push-service.war` file is relative to the `MobileFirstServer` directory that contains the `mfp-ant-deployer.jar` file.

id

The **id** attribute distinguishes different deployments of the push service in the same WebSphere Application Server cell. Without this **id** attribute, two WAR files with the same context roots might conflict and these files would not be deployed.

warFile

Use the **warFile** attribute to specify a different directory for the push service WAR file. You can specify the name of this WAR file with an absolute path or a relative path.

The **installmobilefirstpush**, **updatemobilefirstpush**, and **uninstallmobilefirstpush** Ant tasks support the following elements:

Table 6-91. Inner elements for the **installmobilefirstpush**, **updatemobilefirstpush**, *and* **uninstallmobilefirstpush** *Ant tasks*

Element	Description	Count
`<applicationserver>`	The application server.	1
`<analytics>`	The Analytics.	0..1
`<authorization>`	The authorization server for authenticating the communication with other MobileFirst Server components.	1
`<database>`	The databases.	1
`<property>`	The properties.	0..∞

To specify the authorization server

The `<authorization>` element collects information to configure the authorization server for the authentication communication with other MobileFirst Server components. This element has the following attributes:

Table 6-92. Attributes of the `<authorization>` element

Attribute	Description	Required	Default value
auto	To indicate whether the authorization server URL is computed. The possible values are `true` or `false`.	Required on a WebSphere Application Server Network Deployment cluster or node.	`true`
authorizationURL	The URL of the authorization server.	If mode is not auto.	The context root of the runtime on the local server.
runtimeContextRoot	The context root of the runtime.	No	`/mfp`
pushClientID	The push service confidential ID in the authorization server.	Yes	None
pushClientSecret	The push service confidential client password in the authorization server.	Yes	None

auto If the value is set to true, the URL of the authorization server is computed automatically by using the context root of the runtime on the local application server. The auto mode is not supported if you deploy on WebSphere Application Server Network Deployment on a cluster.

authorizationURL
The URL of the authorization server. If the authorization server is the MobileFirst runtime, the URL is the URL of the runtime. For example: `http://myHost:9080/mfp`.

runtimeContextRoot
The context root of the runtime that is used to compute the URL of the authorization server in the automatic mode.

pushClientID
The ID of this push service instance as a confidential client of the authorization server. The ID and the secret must be registered for the authorization server. It can be registered by **installmobilefirstadmin** Ant task, or from MobileFirst Operations Console.

pushClientSecret
The secret key of this push service instance as a confidential client of the authorization server. The ID and the secret must be registered for the authorization server. It can be registered by **installmobilefirstadmin** Ant task, or from MobileFirst Operations Console.

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

Table 6-93. Attributes for the `<property>` element

Attribute	Description	Required	Default value
name	The name of the property.	Yes	None

Table 6-93. Attributes for the `<property>` element (continued)

Attribute	Description	Required	Default value
`value`	The 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 push service WAR file.

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server push service" on page 6-186.

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:

Table 6-94. Inner elements of the `<applicationserver>` element

Element	Description	Count
`<websphereapplicationserver>` or `<was>`	The parameters for WebSphere Application Server. The `<websphereapplicationserver>` element (or `<was>` in its short form) denotes a WebSphere Application Server instance. WebSphere Application Server full profile (Base, and Network Deployment) are supported, so is WebSphere Application Server Liberty Core and WebSphere Application Server Liberty Network Deployment.	0..1
`<tomcat>`	The parameters for Apache Tomcat.	0..1

The attributes and inner elements of these elements are described in Table 6-105 on page 6-295 through Table 6-114 on page 6-298 of "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

However, for the inner element of the `<was>` element for Liberty collective, see the following table:

Table 6-95. Inner element of the `<was>` element for Liberty collective

Element	Description	Count
`<collectiveMember>`	A Liberty collective member.	0..1

The `<collectiveMember>` element has the following attributes:

Table 6-96. Attributes of the `<collectiveMember>` element

Attribute	Description	Required	Default value
`serverName`	The name of the collective member.	Yes	None
`clusterName`	The cluster name that the collective member belongs to.	Yes	None

Note: If the push service and the runtime components are installed in the same collective member, then they must have the same cluster name. If these components are installed on distinct members of the same collective, the cluster names can be different.

To specify Analytics

The `<analytics>` element indicates that you want to connect the MobileFirst push service to an already installed MobileFirst Analytics service. It has the following attributes:

Table 6-97. Attributes of the `<analytics>` element

Attribute	Description	Required	Default
`install`	To indicate whether to connect the push service to MobileFirst Analytics.	No	`false`
`analyticsURL`	The URL of MobileFirst Analytics services.	Yes	None
`username`	The user name.	Yes	None
`password`	The password.	Yes	None
`validate`	To validate whether MobileFirst Analytics Console is accessible or not.	No	`true`

`install`

Use the **`install`** attribute to indicate that this push service must be connected and send events to MobileFirst Analytics. Valid values are `true` or `false`.

`analyticsURL`

Use the `analyticsURL` attribute to specify the URL that is exposed by MobileFirst Analytics, which receives incoming analytics data.

For example: `http://`*`<hostname>`*`:`*`<port>`*`/analytics-service/rest`

`username`

Use the **`username`** attribute to specify the user name that is used if the data entry point for the MobileFirst 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 Analytics is protected with basic authentication.

`validate`

Use the **`validate`** attribute to validate whether the MobileFirst Analytics Console is accessible or not, and to check the user name authentication with a password. The possible values are `true`, or `false`.

To specify a connection to the push service database

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

You must declare a single database: `<database kind="Push">`. 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 6-98. Attributes of the `<database>` element

Attribute	Description	Required	Default
`kind`	The kind of database (`Push`).	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 6-118 on page 6-300 through Table 6-128 on page 6-304 in "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

Table 6-99. 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
`<cloudant>`	The parameter for Cloudant databases.	0..1
`<driverclasspath>`	The parameter for JDBC driver class path (relational DBMS only).	0..1

Note: The attributes of the `<cloudant>` element are slightly different from the runtime. For more information, see the following table:

Table 6-100. 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 6-100. Attributes of the `<cloudant>` element (continued)

Attribute	Description	Required	Default
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_push_db`

Ant tasks for installation of MobileFirst runtime environments

Reference information for the **installmobilefirstruntime**, **updatemobilefirstruntime**, and **uninstallmobilefirstruntime** Ant tasks.

Task effects

installmobilefirstruntime

The **installmobilefirstruntime** Ant task configures an application server to run a MobileFirst runtime 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 `/mfp`.
- It deploys the runtime WAR file on the application server.
- It declares data sources and – on WebSphere Application Server full profile – JDBC providers for the runtime.
- It deploys the database drivers in the application server.
- It sets MobileFirst configuration properties through JNDI environment entries.
- Optionally, it sets the MobileFirst JNDI environment entries to configure the application server as a server farm member for the runtime.

updatemobilefirstruntime

The **updatemobilefirstruntime** Ant task updates a MobileFirst runtime that is already configured on an application server. This task updates the runtime WAR file. The file must have the same base name as the runtime WAR file that was previously deployed. Other than that, the task does not change the application server configuration, that is, the web application configuration, data sources, and JNDI environment entries.

uninstallmobilefirstruntime

The **uninstallmobilefirstruntime** Ant task undoes the effects of an earlier **installmobilefirstruntime** 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 are added manually to that application.
- It removes the runtime WAR file from the application server.
- It removes the data sources and – on WebSphere Application Server full profile – the JDBC providers for the runtime.
- It removes the associated JNDI environment entries.

Attributes and elements

The **installmobilefirstruntime**, **updatemobilefirstruntime**, and **uninstallmobilefirstruntime** Ant tasks have the following attributes:

Table 6-101. Attributes for the **installmobilefirstruntime**, **updatemobilefirstruntime**, *and* **uninstallmobilefirstruntime** *Ant tasks*

Attribute	Description	Required	Default
contextroot	The common prefix in URLs to the application (context root).	No	`/mfp`
id	To distinguish different deployments.	No	Empty
environmentId	To distinguish different MobileFirst environments.	No	Empty
warFile	The WAR file for MobileFirst runtime.	No	The `mfp-server.war` file is in the same directory as the `mfp-ant-deployer.jar` file.
wasStartingWeight	The start order for WebSphere Application Server. Lower values start first.	No	2

contextroot and id

The **contextroot** and **id** attributes distinguish different MobileFirst projects.

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 service 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 `<installmobilefirstadmin>` invocation, for the administration service application.

warFile

Use the **warFile** attribute to specify a different directory for the MobileFirst runtime WAR file. You can specify the name of this WAR file with an absolute path or a relative path.

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 Server administration service 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.

The **installmobilefirstruntime**, **updatemobilefirstruntime**, and **uninstallmobilefirstruntime** tasks support the following elements:

Table 6-102. Inner elements for the **installmobilefirstruntime**, **updatemobilefirstruntime**, *and* **uninstallmobilefirstruntime** *Ant tasks*

Element	Description	Count
`<property>`	The properties.	0..∞
`<applicationserver>`	The application server.	1
`<database>`	The databases.	1
`<analytics>`	The Analytics.	0..1

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

Table 6-103. Attributes for the `<property>` element.

Attribute	Description	Required	Default value
name	The name of the property.	Yes	None
value	The value for the property.	Yes	None

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 6-104. 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

The `<websphereapplicationserver>` element (or `<was>` in its short form) denotes a WebSphere Application Server instance. WebSphere Application Server full profile (Base, and Network Deployment) are supported, so is WebSphere Application Server Liberty Core and WebSphere Application Server Liberty Network Deployment. The `<websphereapplicationserver>` element has the following attributes:

Table 6-105. 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 6-105. Attributes for the `<websphereapplicationserver>` or `<was>` element (continued)

Attribute	Description	Required	Default
libertyEncoding	The algorithm to encode data source passwords for WebSphere Application Server Liberty. The possible values are `none`, `xor`, and `aes`. Whether the `xor` or `aes` encoding is used, the clear password is passed as argument to the `securityUtility` 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`
jeeVersion	For Liberty profile. To specify whether to install the features of the JEE6 web profile or the JEE7 web profile. Possible values are `6`, `7`, or `auto`.	No	`auto`
configureFarm	For WebSphere Application Server Liberty, and WebSphere Application Server full profile (not for WebSphere Application Server Network Deployment edition and Liberty collective). To specify whether the server is a server farm member. Possible values are `true` or `false`.	No	`false`
farmServerId	A string that uniquely identify a server in a server farm. The MobileFirst Server administration services and all the MobileFirst runtimes that communicate with it must share the same value.	Yes	None

It supports the following element for single-server deployment:

Table 6-106. Inner element 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 attribute:

Table 6-107. The attribute of `<server>` element (single-server deployment)

Attribute	Description	Required	Default
name	The server name.	Yes	None

It supports the following elements for Liberty collective:

Table 6-108. Inner element of the `<was>` element for Liberty collective

Element	Description	Count
`<collectiveMember>`	A Liberty collective member.	0..1

The `<collectiveMember>` element has the following attributes:

Table 6-109. Attributes of the `<collectiveMember>` element

Attribute	Description	Required	Default value
`serverName`	The name of the collective member.	Yes	None
`clusterName`	The cluster name that the collective member belongs to.	Yes	None
`serverId`	A string that uniquely identifies the collective member.	Yes	None
`controllerHost`	The name of the collective controller.	Yes	None
`controllerHttpsPort`	The HTTPS port of the collective controller.	Yes	None
`controllerAdminName`	The administrative user name that is defined in the collective controller. This is the same user that is used to join new members to the collective.	Yes	None
`controllerAdminPassword`	The administrative user password.	Yes	None
`createControllerAdmin`	To indicate whether the administrative user must be created in the basic registry of the collective member. Possible values are `true` or `false`.	No	`true`

It supports the following elements for Network Deployment:

Table 6-110. Inner elements for the `<was>` element (network deployment)

Element	Description	Count
`<cell>`	The entire cell.	0..1
`<cluster>`	All the servers of a cluster.	0..1
`<node>`	All the 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 attribute:

Table 6-111. Attribute for the `<cluster>` element (network deployment)

Attribute	Description	Required	Default
`name`	The cluster name.	Yes	None

The `<node>` element has the following attribute:

Table 6-112. Attribute for the <node> element (network deployment)

Attribute	Description	Required	Default
name	The node name.	Yes	None

The <server> element, which is used in a Network Deployment context, has the following attributes:

Table 6-113. Attributes for the <server> element (network deployment)

Attribute	Description	Required	Default
nodeName	The node name.	Yes	None
serverName	The server name.	Yes	None

The <tomcat> element denotes an Apache Tomcat server. It has the following attribute:

Table 6-114. Attribute of the <tomcat> element

Attribute	Description	Required	Default
installdir	The installation directory of Apache Tomcat. 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
configureFarm	To specify whether the server is a server farm member. Possible values are true or false.	No	false
farmServerId	A string that uniquely identify a server in a server farm. The MobileFirst Server administration services and all the MobileFirst runtimes that communicate with it must share the same value.	Yes	None

The <database> element specifies what information is necessary to access a particular database. The <database> element is specified like the **configuredatabase** Ant task, except that it does not have the <dba> and <client> elements. However, it might have <property> elements. The <database> element has the following attributes:

Table 6-115. Attributes of the <database> element

Attribute	Description	Required	Default
kind	The kind of database (MobileFirstRuntime).	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 6-116. Inner elements for the `<database>` element

Element	Description	Count
`<derby>`	The parameters for Derby.	0..1
`<db2>`	The parameters for DB2.	0..1
`<mysql>`	The parameters for MySQL.	0..1
`<oracle>`	The parameters for Oracle.	0..1
`<driverclasspath>`	The JDBC driver class path.	0..1

The `<analytics>` element indicates that you want to connect the MobileFirst runtime to an already installed MobileFirst Analytics console and services. It has the following attributes:

Table 6-117. Attributes of the `<analytics>` element

Attribute	Description	Required	Default
`install`	To indicate whether to connect the MobileFirst runtime to MobileFirst Analytics.	No	`false`
`analyticsURL`	The URL of MobileFirst Analytics services.	Yes	None
`consoleURL`	The URL ofMobileFirst Analytics Console.	Yes	None
`username`	The user name.	Yes	None
`password`	The password.	Yes	None
`validate`	To validate whether MobileFirst Analytics Console is accessible or not.	No	`true`
`tenant`	The tenant for indexing data that is collected from a MobileFirst runtime.	No	Internal identifier

`install`

Use the **`install`** attribute to indicate that this MobileFirst runtime must be connected and send events to MobileFirst Analytics. Valid values are `true` or `false`.

`analyticsURL`

Use the `analyticsURL` attribute to specify the URL that is exposed by MobileFirst Analytics, which receives incoming analytics data.

For example: `http://`*`<hostname>`*`:`*`<port>`*`/analytics-service/rest`

`consoleURL`

Use the **`consoleURL`** attribute to the URL that is exposed by MobileFirst Analytics, which links to the MobileFirst Analytics console.

For example: `http://<hostname>:<port>/analytics/console`

`username`

Use the **`username`** attribute to specify the user name that is used if the data entry point for the MobileFirst 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 Analytics is protected with basic authentication.

validate

Use the **validate** attribute to validate whether the MobileFirst Analytics Console is accessible or not, and to check the user name authentication with a password. The possible values are `true`, or `false`.

tenant

For more information about this attribute, see "Configuration properties" on page 11-15.

To specify an Apache Derby database

The `<derby>` element has the following attributes:

Table 6-118. Attributes of the `<derby>` element

Attribute	Description	Required	Default
database	The database name.	No	`MFPDATA`, `MFPADM`, `MFPCFG`, `MFPPUSH`, or `APPCNTR`, depending on kind.
datadir	The directory that contains the databases.	Yes	None
schema	The schema name.	No	`MFPDATA`, `MFPCFG`, `MFPADMINISTRATOR`, `MFPPUSH`, or `APPCENTER`, depending on kind.

The `<derby>` element supports the following element:

Table 6-119. Inner element for the `<derby>` element

Element	Description	Count
`<property>`	The data source property or JDBC connection property.	0..∞

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 `mfp-ant-deployer.jar` file is used within the installation directory of IBM MobileFirst Platform Foundation, a `<driverclasspath>` element is not necessary.

To specify a DB2 database

The `<db2>` element has the following attributes:

Table 6-120. Attributes of the `<db2>` element

Attribute	Description	Required	Default
database	The database name.	No	`MFPDATA`, `MFPADM`, `MFPCFG`, `MFPPUSH`, or `APPCNTR`, depending on kind.
server	The host name of the database server.	Yes	None
port	The port on the database server.	No	50000
user	The 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 "Database users and privileges" on page 6-64.	Yes	None
password	The password for accessing databases.	No	Queried interactively
schema	The 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 6-121. Inner element for the `<db2>` element

Element	Description	Count
`<property>`	The 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 6-122. Attributes of the `<mysql>` element

Attribute	Description	Required	Default
database	The database name.	No	`MFPDATA`, `MFPADM`, `MFPCFG`, `MFPPUSH`, or `APPCNTR`, depending on kind.
server	The host name of the database server.	Yes	None

Table 6-122. Attributes of the `<mysql>` element (continued)

Attribute	Description	Required	Default
`port`	The port on the database server.	No	3306
`user`	The 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 "Database users and privileges" on page 6-64.	Yes	None
`password`	The 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 6-123. Alternative attributes for the `<mysql>` element

Attribute	Description	Required	Default
`url`	The URL for connection to the database.	Yes	None
`user`	The 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 "Database users and privileges" on page 6-64.	Yes	None
`password`	The 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 6-124. Inner element for the `<mysql>` element

Element	Description	Count
`<property>`	The 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 6-125. Attributes of the `<oracle>` element

Attribute	Description	Required	Default
database	The database name, or Oracle service name. **Note:** You must always use a service name to connect to a PDB database.	No	*ORCL*
server	The host name of the database server.	Yes	None
port	The port on the database server.	No	1521
user	The 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 "Database users and privileges" on page 6-64. See the note under this table.	Yes	None
password	The 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 **installmobilefirstruntime** Ant task does not convert lowercase letters to uppercase letters in the user name. If the **installmobilefirstruntime** 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 6-126. Alternative attributes of the `<oracle>` element

Attribute	Description	Required	Default
url	The URL for connection to the database.	Yes	None
user	The 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 "Database users and privileges" on page 6-64. See the note under this table.	Yes	None
password	The 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 **installmobilefirstruntime** Ant task does not convert lowercase letters to uppercase letters in the user name. If the **installmobilefirstruntime** 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 element:

Table 6-127. Inner element for the <oracle> element

Element	Description	Count
<property>	The 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 6-128. Attributes for the <property> element in a database-specific element

Attribute	Description	Required	Default
name	The name of the property.	Yes	None
type	Java type of the property values, usually java.lang.String/Integer/Boolean.	No	java.lang.String
value	The value for the property.	Yes	None

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.
- 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 6-129. 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 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 6-130. 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 6-131. 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.

Table 6-132. Inner elements for the <applicationserver> element

Element	Description	Count
websphereapplicationserver or **was**	The parameters for WebSphere Application Server. The <websphereapplicationserver> element (or <was> in its short form) denotes a WebSphere Application Server instance. WebSphere Application Server full profile (Base, and Network Deployment) are supported, so is WebSphere Application Server Liberty Core. Liberty collective is not supported for Application Center.	0..1
tomcat	The parameters for Apache Tomcat.	0..1

The attributes and inner elements of these elements are described in tables Table 6-105 on page 6-295 to Table 6-114 on page 6-298 of the page "Ant tasks for installation of MobileFirst runtime environments" on page 6-293.

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 6-133. 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 6-118 on page 6-300 to Table 6-128 on page 6-304 in "Ant tasks for installation of MobileFirst runtime environments" on page 6-293

Table 6-134. 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

Table 6-134. Inner elements for the <database> element (continued)

Element	Description	Count
`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 6-135. 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 Analytics

The **installanalytics**, **updateanalytics**, and **<uninstallanalytics>** Ant tasks are provided for the installation of MobileFirst Analytics.

About these Ant tasks

The purpose of these Ant Tasks is to configure the MobileFirst Analytics console and the MobileFirst Analytics service with the appropriate storage for the data on an application server.

The task installs MobileFirst Analytics nodes that act as a master and data. For more information, see "Cluster management and Elasticsearch" on page 11-19.

Task effects

<installanalytics>

The `<installanalytics>` Ant task configures an application server to run IBM MobileFirst Analytics. This task has the following effects:

- It deploys the MobileFirst Analytics Service and the MobileFirst Analytics Console WAR files on the application server.
- It declares the MobileFirst Analytics Service web application in the specified context root `/analytics-service`.
- It declares the MobileFirst Analytics Console web application in the specified context root `/analytics`.
- It sets MobileFirst Analytics Console and MobileFirst 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 Analytics Console.

`<updateanalytics>`

The `<updateanalytics>` Ant task updates the already configured MobileFirst Analytics Service and MobileFirst 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 Analytics Service and the MobileFirst Analytics Console web applications with their respective context roots.
- It removes the MobileFirst Analytics Service and the MobileFirst 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 6-136. Attributes for the <installanalytics>, <updateanalytics>, and <uninstallanalytics> Ant tasks

Attribute	Description	Required	Default
serviceWar	The WAR file for the MobileFirst 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 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 6-137. Inner elements for the <installanalytics>, <updateanalytics>, and <uninstallanalytics> Ant tasks

Attribute	Description	Required	Default
console	MobileFirst 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

Table 6-137. Inner elements for the <installanalytics>, <updateanalytics>, and <uninstallanalytics> Ant tasks (continued)

Attribute	Description	Required	Default
property	Properties.	No	0..∞

To specify a MobileFirst Analytics Console

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

Table 6-138. 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 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 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 Analytics Console in a browser.
- `analytics-console.sh`: This file is a UNIX shell script. It opens the MobileFirst Analytics Console in a browser.

Note: These shortcuts do not include the ElasticSearch tenant parameter.

The `<console>` element supports the following nested element:

Table 6-139. 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 6-140. 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 6-141. 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:
- `mfpmonitor`
- `mfpoperator`
- `mfpdeployer`
- `mfpadmin`

To specify a type of storage for MobileFirst Analytics

The `<storage>` element indicates which underlying type of storage MobileFirst Analytics uses to store the information and data it collects.

It supports the following element:

Table 6-142. Inner element of the <storage> element

Element	Description	Count
elasticsearch	ElasticSearch cluster	1

The `<elasticsearch>` element collects the parameters about an ElasticSearch cluster.

Table 6-143. 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

Table 6-143. Attributes of the <elasticsearch> element (continued)

Attribute	Description	Required	Default
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 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 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 6-293.

Table 6-144. 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 6-145. Inner element of the <property> element

Element	Description	Count
property	Properties	0..∞

By using this element, you can define your own JNDI properties.

This element has the following attributes:

Table 6-146. 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

Learn about runtime database tables, their purpose, and order of magnitude of data stored in each table. In relational databases, the entities are organized in database tables.

Database used by MobileFirst Server runtime

The following table provides a list of runtime database tables, their descriptions, and how they are used in relational databases.

Table 6-147. Common runtime database tables

Relational database table name	Description	Order of magnitude
LICENSE_TERMS	Stores the various license metrics captured every time the device decommissioning task is run.	Tens of rows. This value does not exceed the value set by the JNDI property **mfp.device.decommission.when** property. For more information about JNDI properties, see "List of JNDI properties for MobileFirst runtime" on page 6-183
ADDRESSABLE_DEVICE	Stores the addressable device metrics daily. An entry is also added each time that a cluster is started.	About 400 rows. Entries older than 13 months are deleted daily.
MFP_PERSISTENT_DATA	Stores instances of client applications that have registered with the OAuth server, including information about the device, the application, users associated with the client and the device status.	One row per device and application pair.
MFP_PERSISTENT_CUSTOM_ATTR	Custom attributes that are associated with instances of client applications. Custom attributes are application-specific attributes that were registered by the application per each client instance.	Zero or more rows per device and application pair
MFP_TRANSIENT_DATA	Authentication context of clients and devices	Two rows per device and application pair; if using device single sign-on an extra two rows per device. For more information about SSO, see "Configuring device single sign-on (SSO)" on page 7-301.

Table 6-147. Common runtime database tables (continued)

Relational database table name	Description	Order of magnitude
SERVER_VERSION	The product version.	One row

Database used by MobileFirst Server administration service

The following table provides a list of administration database tables, their descriptions, and how they are used in relational databases.

Table 6-148. Common administration database tables.

Relational database table name	Description	Order of Magnitude
ADMIN_NODE	Stores information about the servers that run the administration service. In a stand-alone topology with only one server, this entity is not used.	One row per server; empty if a stand-alone server is used.
AUDIT_TRAIL	Stores an audit trail of all administrative actions performed with the administration service.	Thousands of rows.
CONFIG_LINKS	Stores the links to the live update service. Adapters and applications might have configurations that are stored in the live update service, and the links are used to find those configurations.	Hundreds of rows. Per adapter, 2-3 rows are used. Per application, 4-6 rows are used.
FARM_CONFIG	Stores the configuration of farm nodes when a server farm is used.	Tens of rows; empty if no server farm is used.
GLOBAL_CONFIG	Stores some global configuration data.	1 row.
PROJECT	Stores the names of the deployed projects.	Tens of rows.
PROJECT_LOCK	Internal cluster synchronization tasks.	Tens of rows.
TRANSACTIONS	Internal cluster synchronization table; stores the state of all current administrative actions.	Tens of rows.
MFPADMIN_VERSION	The product version.	One row.

Database used by MobileFirst Server live update service

The following table provides a list of live update service database tables, their descriptions, and how they are used in relational databases.

Table 6-149. Live update service tables

Relational database table name	Description	Order of magnitude
CS_SCHEMAS	Stores the versioned schemas that exist in the platform.	One row per schema.
CS_CONFIGURATIONS	Stores instances of configurations for each versioned schema.	One row per configuration
CS_TAGS	Stores the searchable fields and values for each configuration instance.	Row for each field name and value for each searchable field in configuration.
CS_ATTACHMENTS	Stores the attachments for each configuration instance.	One row per attachment.
CS_VERSION	Stores the version of the MFP that created the tables or instances.	Single row in the table with the version of MFP.

Database used by MobileFirst Server push service

The following table provides a list of push service database tables, their descriptions, and how they are used in relational databases.

Table 6-150. Push service tables

Relational database table name	Description	Order of magnitude
PUSH_APPS	Push notification table; stores details of push applications.	One row per application.
PUSH_ENV	Push notification table; stores details of push environments.	Tens of rows.
PUSH_TAGS	Push notification table; stores details of defined tags.	Tens of rows.
PUSH_DEVICES	Push notification table. Stores a record per device.	One row per device.
PUSH_SUBSCRIPTIONS	Push notification table. Stores a record per tag subscription.	One row per device subscription.
PUSH_MESSAGES	Push notification table; stores details of push messages.	Tens of rows.
PUSH_MESSAGE_SEQUENCE_TABLE	Push notification table; stores the generated sequence ID.	One row.
PUSH_VERSION	The product version.	One row.

For more information about setting up the databases, see “Setting up databases” on page 6-63.

Sample configuration files

IBM MobileFirst Platform Foundation includes a number of sample configuration files to help you get started with the Ant tasks to install the MobileFirst Server.

The easiest way to get started with these Ant tasks is by working with the sample configuration files provided in the `MobileFirstServer/configuration-samples/` directory of the MobileFirst Server distribution. For more information about installing MobileFirst Server with Ant tasks, see "Installing with Ant Tasks" on page 6-110

List of sample configuration files

Pick the appropriate sample configuration file. The following files are provided

Table 6-151. Sample configuration files provided with IBM MobileFirst Platform Foundation

Task	Derby	DB2	MySQL	Oracle
Create databases with database administrator credentials	`create-database-derby.xml`	`create-database-db2.xml`	`create-database-mysql.xml`	`create-database-oracle.xml`
Install MobileFirst Server on Liberty	`configure-liberty-derby.xml`	`configure-liberty-db2.xml`	`configure-liberty-mysql.xml` (See Note on MySQL)	`configure-liberty-oracle.xml`
Install MobileFirst Server 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`

Install MobileFirst Server 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`

Install MobileFirst Server on Apache Tomcat	`configure-tomcat-derby.xml`	`configure-tomcat-db2.xml`	`configure-tomcat-mysql.xml`	`configure-tomcat-oracle.xml`

Table 6-151. Sample configuration files provided with IBM MobileFirst Platform Foundation (continued)

Task	Derby	DB2	MySQL	Oracle
Install MobileFirst Server on Liberty collective	Not relevant	`configure-libertycollective-db2.xml`	`configure-libertycollective-mysql.xml`	`configure-libertycollective-oracle.xml`

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.

Sample configuration files for MobileFirst Analytics

IBM MobileFirst Platform Foundation includes a number of sample configuration files to help you get started with the Ant tasks to install the MobileFirst Analytics Services, and the MobileFirst 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 6-152. Sample configuration files provided with IBM MobileFirst Platform Foundation

Task	Application server
Install MobileFirst Analytics Services and Console on WebSphere Application Server Liberty profile	`configure-liberty-analytics.xml`
Install MobileFirst Analytics Services and Console on Apache Tomcat	`configure-tomcat-analytics.xml`
Install MobileFirst Analytics Services and Console on WebSphere Application Server full profile	`configure-was-analytics.xml`

Table 6-152. Sample configuration files provided with IBM MobileFirst Platform Foundation (continued)

Task	Application server
Install MobileFirst Analytics Services and Console on WebSphere Application Server Network Deployment, single server	`configure-wasnd-server-analytics.xml`
Install MobileFirst Analytics Services and Console on WebSphere Application Server Network Deployment, cell	`configure-wasnd-cell-analytics.xml`
Install MobileFirst Analytics Services and Console on WebSphere Application Server Network Deployment, node	`configure-wasnd-node.xml`
Install MobileFirst 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 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.

Step 5

Run the command:

```
ant -f configure-file.xml install
```

This command installs your MobileFirst Analytics Services and MobileFirst Analytics Console components in the application server.

To install updated MobileFirst Analytics Services and MobileFirst 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 Analytics Services and MobileFirst Analytics Console components.

Developing applications

The process for developing applications has steps that are common to all environments: setting up a server, creating an initial server registration and corresponding configuration files, creating a new (or opening an existing) project in your chosen IDE, and adding the necessary SDK files to your IDE project. Also, server-side adapters can be developed as needed for the application.

Each MobileFirst application consists of server-side and client-side development. Before you can initially run your client app and connect to the server resources, the client app needs to be registered to the server.

Setting up the environment

Regardless of the target device platform, all MobileFirst applications need to be set up before they can be developed.

- Set up the IBM MobileFirst Platform Foundation Developer Kit if necessary. For more information, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.
- Create a set of MobileFirst SDK files for adding to your application. For more information, see "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26.
- Develop your client app.
- Register your app to the MobileFirst Development Server, which is installed with the IBM MobileFirst Platform Foundation Developer Kit.
- Add server-side resources (adapters) for your app.

Using package-management tools to add the SDK to your existing app

- To develop a native iOS application, use the MobileFirst iOS SDK with CocoaPods. For more information, see "Adding MobileFirst SDK to an iOS Xcode project using CocoaPods" on page 7-29.
- To develop a native Android application, use the MobileFirst Android SDK using Gradle. For more information, see "Setting up Android Studio projects with Gradle" on page 7-53.
- To develop a native Windows 8 Universal application or a Windows 10 Universal Windows Platform (UWP) application, use the MobileFirst Windows Universal SDK with NuGet. For more information, see "Adding the MobileFirst SDK by using NuGet" on page 7-66.

Developing the app

MobileFirst can be developed for Apple devices using the iOS Objective-C SDK, Android-based devices using the Android Java SDK, web-browser devices using the web JavaScript SDK, and Windows devices using the Windows C# SDK. In addition, SDKs are provided for developing Cordova apps through a combination of a native platform SDK (iOS or Android) and JavaScript.

- For native Android development, see "Developing native applications in Android Studio" on page 7-52.

- For native iOS development, see "Developing native applications for iOS in Xcode" on page 7-27
- For Windows development, see "Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal" on page 7-65.
- For web development, see "Developing web applications" on page 7-73.
- For Cordova development, see "Developing Cordova applications" on page 7-83.

Using server-side resources

When the client app can connect to the server, it can use server-side resources such as adapters and security.
- Adapters can be developed in Java or JavaScript. See "Developing the server side of a MobileFirst application" on page 7-187.
- The app can be secured in a number of ways. See "MobileFirst security framework" on page 7-265.

Build and deployment

For more information about building and deploying applications to a test or production server, see "Deploying MobileFirst applications to test and production environments" on page 10-2.

Development concepts and overview

When you develop your app with MobileFirst tools, you must develop or configure a variety of components and elements. Learning about the components and elements involved when developing your app helps your development proceed smoothly.

Applications

Applications are built for a target MobileFirst Server. They have a server-side configuration on the target server. You must register your applications on the MobileFirst Server before you can configure them.

In IBM MobileFirst Platform Foundation, applications are identified by the following elements:
- An app ID
- A version number
- A target deployment platform

Note: The version number is not applicable to web applications. You cannot have multiple versions of the same web application.
These identifiers are used on both the client side and the server side to ensure that apps are deployed correctly and use only resources assigned to them. Different parts of IBM MobileFirst Platform Foundation use various combinations of these identifiers in different ways.

The app ID depends on the target deployment platform:

Android
For Android apps, the identifier is the application package name.

iOS For iOS apps, the identifier is the application bundle ID.

Windows
For Windows apps, the identifier is application assembly name.

Web For web apps, the identifier is a unique ID that is assigned by the developer.

If apps for different target platforms all have the same app ID, then IBM MobileFirst Platform Foundation considers all of these apps to be the same app with different platform instances. For example, the following apps are considered to be different platform instances of the same app:

- An iOS app with a bundle ID of `com.mydomain.mfp`
- An Android app with a package name of `com.mydomain.mfp`
- A Windows 10 Universal Windows Platform app with an assembly name of `com.mydomain.mfp`
- A web app with an assigned ID of `com.mydomain.mfp`.

The target deployment platform for the app is independent of whether the app was developed as a native app or as a Cordova app. For example, the following apps are both considered to be iOS apps in IBM MobileFirst Platform Foundation:

- An iOS app that you develop with Xcode and native code
- An iOS app that you develop with Cordova cross-platform development technologies

Application configuration

An application is configured on both the client side and the server side. For native and Cordova iOS, Android, and Windows applications, the client configuration is stored in a client properties file (`mfpclient.plist` for iOS, `mfpclient.properties` for Android, or `mfpclient.resw` for Windows). For web applications, the configuration properties are passed as parameters to the SDK initialization method (see "Initializing the MobileFirst SDK" on page 7-79). The client configuration properties include the application ID and information such as the URL of the MobileFirst Server runtime and security keys that are required to access to the server. The server configuration for the app includes information like app management status, web resources for Direct Update,configured security scopes, and log configuration.

The client configuration must be defined before you build the application. The client-app configuration properties must match the properties that are defined for this app in the MobileFirst Server runtime. For example, security keys in the client configuration must match the keys on the server. For non-web apps, you can change the client configuration with the MobileFirst Platform CLI (**`mfpdev`** command).

The server configuration for an app is tied to the combination of app ID, version number, and target platform. The version number is not applicable to web apps. You must register your app to a MobileFirst Server runtime before you can add server-side configurations for the app.

Configuring the server side of an app is typically done with the MobileFirst Operations Console. You can also configure the server side of an app with the following methods:

- Grab existing JSON configuration files from the server with the **`mfpdev app pull`** command, update the file, and upload the changed configuration with the **`mfpdev app push`** command.

- Use the **mfpadm** program or Ant task.

  For information about using **mfpadm**, see "Administering MobileFirst applications through the command line" on page 10-47 and "Administering MobileFirst applications through Ant" on page 10-23.
- Use the REST API of the MobileFirst administration service.

  For information about the REST API, see "REST API for the MobileFirst Server administration service" on page 8-7

You can also use these methods to automate configuring your MobileFirst Server.

Remember: You can modify the server configuration even while a MobileFirst Server is running and receiving traffic from apps. You do not need to stop the server to change the server configuration for an app.

On a production server, the app version typically corresponds to the version of the application published to an app store. Some server configuration elements like the configuration for app authenticity, are unique to the app published to the store.

MobileFirst Server

The server side of your mobile app is MobileFirst Server. MobileFirst Server gives you access to features like application management and application security, as well giving your mobile app secure access to your other backend systems through adapters.

MobileFirst Server is the core component that delivers many IBM MobileFirst Platform Foundation features, including the following features:

- Application management
- Application security, including authenticating devices and users and verifying application authenticity
- Secure access to backend services through adapters
- Updating Cordova app Web resources with Direct Update
- Push notifications and push subscriptions
- App analytics

You need to use MobileFirst Server throughout your app's lifecycle from development and test through to production deployment and maintenance. A preconfigured server is available for you to use when you develop your app. For information about the MobileFirst Development Server to use when you develop your app, see "Setting up the MobileFirst Development Server" on page 7-12.

MobileFirst Server consists of the following components. All of these components are also included in the MobileFirst Development Server. In simple cases, they are all running on the same application server, but in a production or test environment, the components can be run on different application servers. For information about possible topologies for these MobileFirst Server components, see "Topologies and network flows" on page 6-78.

MobileFirst Operations Console and the MobileFirst Server administration service

The operations console is a web interface that you can use to view and edit the MobileFirst Server configurations. You can also access the MobileFirst Analytics Console from here.

The context root for the operations console in the development server is `/mfpconsole`.

The administration service is the main entry point for managing your apps. You can access the administration service through a web-based interface with the MobileFirst Operations Console. You can also access the administration service with the **`mfpadm`** command-line tool or the administration service REST API.

MobileFirst runtime

The runtime is the main entry point for a MobileFirst client app. The runtime is also the default authorization server for the IBM MobileFirst Platform Foundation OAuth implementation.

In advanced and rare cases, you can have multiple instances of a device runtime in a single MobileFirst Server. Each instance has its own context root. The context root is used to display the name of a runtime in the operations console. Use multiple instances in cases where you require different server-level configuration such as secret keys for keystore.

If you have only one instance of a device runtime in MobileFirst Server, you do not typically need to know the runtime context root. For example, when you register an application to a runtime with the **`mfpdev app register`** command when the MobileFirst Server has only one runtime, the application is registered automatically to that runtime.

MobileFirst Server push service

The push service is your main access point for push-related operations like push notifications and push subscriptions. To contact the push services, client apps use the URL of the runtime but replace the context root with `/mfppush`. You can configure and manage the push service with the MobileFirst Operations Console or the push service REST API.

If you run the push services in a separate application server from the MobileFirst runtime, you must route the push service traffic to the correct application server with your HTTP server.

MobileFirst Analytics and the MobileFirst Analytics Console

IBM MobileFirst Analytics is an optional component that provides a scalable analytics feature that you can access from the MobileFirst Operations Console. This analytics feature lets you search for patterns, problems and platform usage statistics across logs and events that are collected from devices, apps, and servers.

From the MobileFirst Operations Console, you can define filters to enable or disable data forwarding to the analytics service. You can also filter the type of information that is sent. On the client side, you can use the client-side log capture API to send events and data to the analytics server. For more information about the client-side log capture API, see “Logger SDK” on page 11-37.

After you install and configure MobileFirst Server into the topology that you want, any further configuration of MobileFirst Server and its applications can be done entirely through any of the following methods:

- The MobileFirst Operations Console
- The MobileFirst Server administration service REST API
- The **`mfpadm`** command-line tool

After the initial installation and configuration, you do not need to access any application server console or interface to configure IBM MobileFirst Platform Foundation.

When you deploy your app to production, you can deploy your app to the following MobileFirst Server production environments:

- On-premises.

  For information about installing and configuring MobileFirst Server for your on-premises environment, see "Installing IBM MobileFirst Platform Server" on page 6-2.
- On the cloud

  For information, see "Deploying MobileFirst Server to the cloud" on page 9-1

Adapters

Adapters in IBM MobileFirst Platform Foundation securely connect your back-end systems to client applications and cloud services.

You can write adapters in either JavaScript or Java, and you can build and deploy adapters as Maven projects. Adapters are deployed to a MobileFirst runtime in MobileFirst Server.

In a production system, adapters typically run in a cluster of application servers. Implement your adapters as REST services with no session information and stored locally on the server to ensure that your adapter works well in a clustered environment.

An adapter can have user-defined properties. These properties can be configured on the server side without redeploying the adapter. For example, you can change the URL that your adapter uses to access resources when you move from test to production.

You can deploy an adapter to a MobileFirst runtime from the MobileFirst Operations Console, by using the **`mfpdev adapter deploy`** command, or directly from Maven.

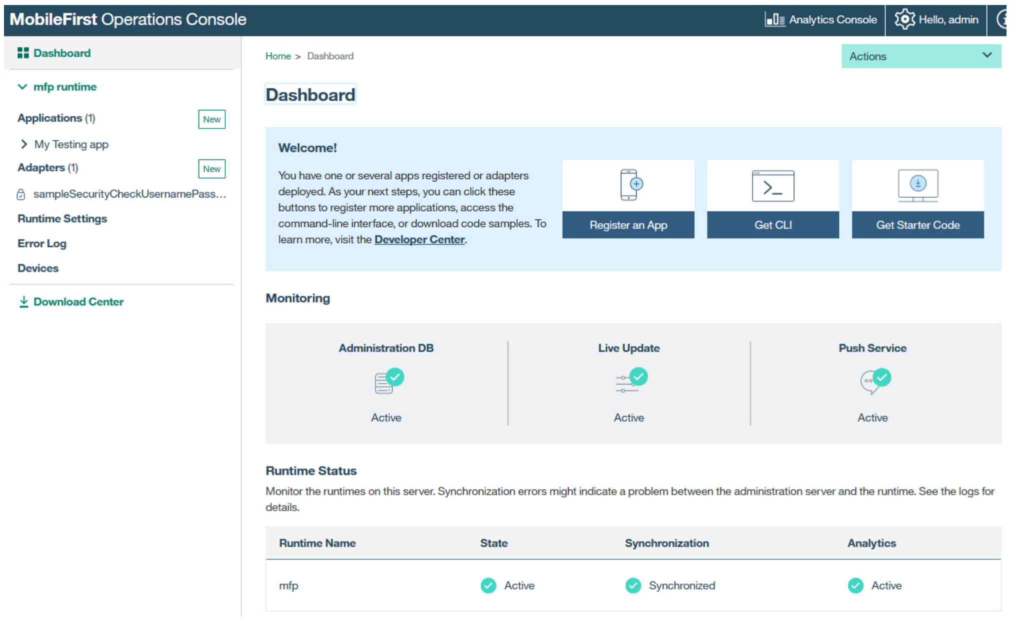
For more information about adapters, see Adapters overview.

MobileFirst Operations Console overview

The MobileFirst Operations Console is a web application that provides a graphical user interface to simplify some MobileFirst development and administration tasks.

You can open the console from a browser. If you have installed IBM MobileFirst Platform Command Line Interface (CLI), you can also open the console by running the **`mfpdev server console`** command. For more information, see "Opening the MobileFirst Operations Console" on page 7-12.

To access the console, you need a valid user name and password, which is assigned to you by your MobileFirst Server administrator according to your role. The MobileFirst Development Server is preconfigured with the `admin/admin` combination.



You have one or several apps registered or adapters deployed. As your next steps, you can click these buttons to register more applications, access the command-line interface, or download code samples. To learn more, visit the Developer Center.

Monitor the runtimes on this server. Synchronization errors might indicate a problem between the administration server and the runtime. See the logs for details.

Runtime Name	State	Synchronization	Analytics
mfp	✓ Active	✓ Synchronized	✓ Active

Quick tour

The console is designed for development, deployment, management, and monitoring tasks.

As a developer

- Develop applications for any environment and register them to MobileFirst Server.
- See all your deployed applications and adapters at a glance. See the Dashboard.
- Manage and configure registered applications, including remote disablement, Direct Update, and security configurations for application authenticity and user authentication.
- Set up push notification by deploying certificates, creating notification tags, and sending notification.
- Create and deploy adapters.
- Download samples.

As an IT administrator

- Monitor various services.
- Search for devices that access MobileFirst Server and manage their access rights.
- Update adapter configurations dynamically.
- Adjust client logger configurations through log profiles.
- Track how product licenses are used.

Getting Started

To get started with the MobileFirst Operations Console, see the Using the MobileFirst Platform Operations Console tutorial.

See also

For more information about the command-line interface for development, see "The MobileFirst command-line interface (CLI)" on page 7-13.

For more information about users and roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

Client app development environments

With IBM MobileFirst Platform Foundation, you can develop your mobile app in the development environment of your choice as a native app or as a Cordova app.

You need two pieces in your development environment to develop a MobileFirst client application: a MobileFirst SDK, and the MobileFirst Platform CLI (**`mfpdev`**).

Depending on the target platform, the MobileFirst SDK is available in different ways:

Native iOS apps

You can add the MobileFirst SDK to Xcode by using CocoaPods, or you can set up your development environment manually.

For more information about setting up your iOS development environment, see "Developing native applications for iOS in Xcode" on page 7-27.

Native Android apps

You can add the MobileFirst SDK to Android Studio with Gradle, or you can set up your development environment manually.

For more information about setting up your Android development environment, see "Developing native applications in Android Studio" on page 7-52.

Native Windows apps

You can add the MobileFirst SDK to VisualStudio with NuGet, or you can set up your development environment manually.

For more information about setting up your Windows development environment, see "Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal" on page 7-65.

Web apps

You can acquire the MobileFirst web SDK either from the IBM MobileFirst Platform Operations Console or by using **`npm`** (node package manager), and then manually add the SDK to your web project.

For more information about setting up your web development environment, see "Developing web applications" on page 7-73.

Cordova apps

The MobileFirst SDK for Cordova is available as a plug-in that you can obtain by using **`npm`** (node package manager).

You can use IBM MobileFirst Studio plug-in to develop Cordova apps. For more information, see "IBM MobileFirst Studio plug-in for managing Cordova projects in Eclipse" on page 7-114.

For more information about setting up your environment to develop MobileFirst apps with Cordova, see "Prerequisites for developing Cordova apps with MobileFirst features" on page 7-84.

In addition to these development-platform specific installation methods, you can also get the IBM MobileFirst Platform Foundation Developer Kit to give you all of the components that you need to start developing your MobileFirst app.

When you develop your app, use the MobileFirst Platform CLI for the following types of tasks:
- Configuring the client side of the application
- Updating the server-side configuration of the application
- Defining the target
- Previewing and updating Cordova apps

For more information about the MobileFirst Platform CLI, see "The MobileFirst command-line interface (CLI)" on page 7-13.

Setting up the development environment

Install MobileFirst Development Server and MobileFirst Platform CLI before you develop the client-side or the server-side of your MobileFirst application.

You need MobileFirst Development Server and MobileFirst Platform CLI for many development tasks. You install these tools with IBM MobileFirst Platform Foundation Developer Kit. The IBM MobileFirst Platform Foundation Developer Kit can also be used to get the MobileFirst SDKs or to download starter code.

Getting started with MobileFirst development

To develop an application for any of the supported platforms, you follow the same general pattern. You go through several basic steps to develop your application: start the MobileFirst Development Server, open the MobileFirst Operations Console, customize sample code, and create an adapter. Tutorials help you get started.

Before you begin

To install a development server and open the console, see "Setting up the MobileFirst Development Server" on page 7-12.

About this task

For step-by-step development instructions for each of the supported platforms, see the Quick Start tutorials on the Developer Center website or "Getting started with a sample MobileFirst application" on page 7-25.

The IBM MobileFirst Platform Foundation Developer Kit

The IBM MobileFirst Platform Foundation Developer Kit contains all you need to start developing and testing MobileFirst applications. The kit includes many components, such as:

- A version of MobileFirst Server, called the MobileFirst Development Server.
- The IBM MobileFirst Platform Command Line Interface (CLI).
- The migration assistance tool.
- Sample applications.

Installing the IBM MobileFirst Platform Foundation Developer Kit

Before you begin

- You must have IBM MobileFirst Platform Foundation V8.0.0.
- You must have computer with a supported OS X, Windows, or Linux operating system installed. See "System requirements" on page 2-7 for more information about supported operating systems.
- To install the IBM MobileFirst Platform Command Line Interface (CLI), you must have Node.js 4.0.0 or later installed. If you do not have internet access, it must be preinstalled. Otherwise, you can install Node.js as part of this procedure, but internet access is required to do so.

Note: If you need to set up your development environment on a computer that has no internet access, you can install components offline. See How to set up an offline IBM MobileFirst development environment.

About this task

The IBM MobileFirst Platform Foundation Developer Kit is available for download as a compressed file for your operating system. You download the file, uncompress it, then install its components.

Procedure

1. Download the IBM MobileFirst Platform Foundation Developer Kit from the Download page.

   You download one of the following files, depending on your operating system:
   - OS X:`mfp-devkit-mac-`*`dddd`*`-`*`tttt`*`.zip`
   - Linux: `mfp-devkit-linux-`*`dddd`*`-`*`tttt`*`.bin`
   - Windows: `mfp-devkit-windows-`*`dddd`*`-`*`tttt`*`.exe`

   where *dddd* is a date stamp and *tttt* is a time stamp.
2. Start the installation program as follows:
   - OS X: Uncompress the `mfp-devkit-mac-`*`dddd`*`-`*`tttt`*`.zip` file and click or double-click to launch the installation program.
   - Linux: Type `./`*`file_name`*`.bin`, where *`file_name`*`.bin` is the file you just downloaded.
   - Windows: Do one of the following:
     - In Windows Explorer, navigate to the directory into which you downloaded the file, and double-click the file.
     - Open a command prompt window and navigate to the directory into which you downloaded the file. Then, type *`file_name`*`.exe`, where *`file_name`*`.exe` is the file you just downloaded.

   This starts the installation program.
3. Follow the installation program prompts. You must accept both IBM and non-IBM terms of the license agreement to continue.
4. Open a command prompt or terminal window, and navigate to the directory in which you installed the IBM MobileFirst Platform Foundation Developer Kit. You should find two subdirectories, `license` and `mfp-server` , a `README.txt` file, and the following executable files for managing the MobileFirst Development Server:
   - `sh` files for OS X and Linux: `run.sh`, `console.sh`, `stop.sh`

- `cmd` files for Windows: `run.cmdconsole.cmd`, `stop.cmd`

5. From the command prompt or terminal window, start the MobileFirst Development Server:
   - For OS X or Linux, enter
     ```
     ./run.sh
     ```
   - For Windows, enter
     ```
     run
     ```

   Starting the server might take a few minutes. When the message `The server mfp is ready to run a smarter planet` is displayed, the server is up and running.

   Important: Do not close the command prompt or terminal window in which you started the server. If you close the window, the server stops running.

   Tip: You can also launch the server as a background process. To launch the server in the background, use the `-bg` option. For example, for OS X: `run.sh -bg`. If you run the server as a background process, you can close the command prompt or terminal window without stopping the server.
6. Start the IBM MobileFirst Platform Operations Console:
   - For OS X or Linux, enter
     ```
     ./console.sh
     ```
   - For Windows, enter
     ```
     console
     ```
7. Log in to the MobileFirst Operations Console. Use the following default login credentials:
   - **User**: `admin`
   - **Password**: `admin`
8. Download and install the IBM MobileFirst Platform Command Line Interface (CLI). To download and install the CLI, follow these steps:
   a. From the MobileFirst Operations Console Dashboard, click **Get Starter Code**.
   b. From the Downloads page, select the **Tools** tab.
   c. Under Developer CLI, click **Download**.
   d. Save the file `mfpdev-cli.tgz` to your local computer.
   e. (Required only if Node.js is not already installed. If Node.js is already installed, skip this step.) Install Node.js, version 4.0.0 or later. To download and install Node.js, click the link **Node.js to be installed** in the console. This takes you to the Node.js web site. Follow the download and installation instructions there.

      Note: Alternatively, you can reach the Node.js web site by clicking this link: Node.js web site
   f. Open a command prompt or terminal window and run the following command:
      ```
      npm install -g --no-optional path_cli_file
      ```
      where *`path_cli_file`* is the full path and name of the downloaded file, including extension. For example, if the file is in the current working directory:
      ```
      npm install -g --no-optional mfpdev-cli.tgz
      ```

9. Optional: You can install the migration assistance tool by repeating step 8 on page 7-11 and substituting the filename for the migration assistance tool (`mfp-migrate-cli.tgz`) for the name of the CLI file (`mfpdev-cli`).
10. Download the IBM MobileFirst Platform Foundation SDK that is appropriate to your target platform. For complete instructions, see “Acquiring the MobileFirst SDK from the MobileFirst Operations Console” on page 7-26.
11. Optional: Obtain sample starter applications. From the IBM MobileFirst Platform Operations Console, download onre or more sample starter application. For complete instructions, see “Getting started with a sample MobileFirst application” on page 7-25.

Results

You have a MobileFirst Development Server installed and running. You have the IBM MobileFirst Platform Command Line Interface (CLI) installed, and (optionally) you have one or more starter applications.

What to do next

To get familiar with MobileFirst development, try the starter app. For an example of using the CLI to perform various development tasks, see “Getting started with the MobileFirst CLI” on page 7-22.

Setting up the MobileFirst Development Server

Learn how to install and use MobileFirst Development Server, a pre-configured MobileFirst Server that you can use for test and development.

MobileFirst Development Server is a preconfigured MobileFirst Server that you can use for test and development. You install MobileFirst Development Server by installing the IBM MobileFirst Platform Foundation Developer Kit. For more information, see “The IBM MobileFirst Platform Foundation Developer Kit” on page 7-9.

Starting the MobileFirst Development Server

Follow this procedure to start the development server.

Before you begin

Make sure that the development server is installed. To install a development server, see “Installing the IBM MobileFirst Platform Foundation Developer Kit” on page 7-10.

Procedure

1. Open a command line window.
2. Go to the server `install` directory.
3. Depending on your system, run the following command:
   - In Mac and Linus, run the command **`./run.sh`**.
   - In Windows, run the command **`run.cmd`**.

Opening the MobileFirst Operations Console

You open the MobileFirst Operations Console by loading a URL to your browser.

Before you begin

Make sure that the server is started. To start a development server that is installed locally, see “Starting the MobileFirst Development Server” on page 7-12.

About this task

Procedure

In a browser window, load this URL: `http://`*`your-server-host`*`:`*`server-port`*`/mfpconsole`.
For a list of supported browsers, see “System requirements” on page 2-7.
For example, if you run the MobileFirst Development Server locally, use `http://localhost:9080/mfpconsole`
To access the console, you need a valid user name and password, which is assigned to you by your MobileFirst Server administrator according to your role. The MobileFirst Development Server is preconfigured with the `admin/admin` combination.

Stopping the MobileFirst Development Server

Follow this procedure to stop the MobileFirst Development Server.

Before you begin

Make sure that the development server is installed. To install a development server, see “Installing the IBM MobileFirst Platform Foundation Developer Kit” on page 7-10.

Procedure

1. Open a command line window.
2. Go to the server `install` directory.
3. Depending on your system, run the following command:
   - In Mac and Linux, run the command **`./stop.sh`**.
   - In Windows, run the command **`stop.cmd`**.

The MobileFirst command-line interface (CLI)

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to develop and manage applications, in addition to using the IBM MobileFirst Platform Operations Console. Some aspects of the MobileFirst development process must be done with the CLI.

The commands, all prefaced with **`mfpdev`**, support the following types of tasks:

- Registering apps with the MobileFirst Server
- Configuring your app
- Creating, building, and deploying adapters
- Previewing and updating Cordova apps

You can use the **`mfpdev`** commands on their own, or in parallel with the MobileFirst Operations Console. You can also use the commands in scripts for automated testing, build, and deployment flows.

The **`mfpdev`** commands have two modes: interactive mode and direct mode. In interactive mode, you enter the command without options, and you are prompted for responses. In direct mode, you enter the full command, including options, and prompts are not provided. When applicable, the prompts are context-sensitive to the target platform of the app, as determined by the directory from which you run the command. Use the up and down arrow keys on your keyboard to move through the selections, and press the **Enter** key when the selection you want is highlighted and preceded by a > character.

Some **mfpdev** commands require connectivity to a MobileFirst Server. If a MobileFirst Server is locally installed and running, these commands automatically detect it and use it as the default server if no other server is explicitly set as the default.

Tip: Another set of commands, the **mfpadm** commands, are available for MobileFirst Server administration. For more information about these commands, see "Administering MobileFirst applications through the command line" on page 10-47.

The IBM MobileFirst Platform Command Line Interface (CLI) (**mfpdev** commands) supports development for all native and hybrid platforms that are supported by IBM MobileFirst Platform Foundation. However, Android projects must be created with Android Studio and Gradle. Android projects that are created with the Eclipse and ADT (Android Developer Toolkit) are not supported by the **mfpdev** CLI. For instructions on how to convert your Eclipse ADT Android project to an Android Studio project, see Migrating from Eclipse ADT at the Android developer web site.

To install the MobileFirst Platform CLI, see "Installing the MobileFirst Platform CLI" on page 7-15.

Prerequisite software for using the CLI

To use the IBM MobileFirst Platform Command Line Interface (CLI), you might need some additional software, depending on your development environment and application development goals.

You can obtain the IBM MobileFirst Platform Command Line Interface (CLI) in two ways:

- It is included in the IBM MobileFirst Platform Foundation Developer Kit. For more information about the IBM MobileFirst Platform Foundation Developer Kit, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.
- You can install it from Node Package manager (npm) or JazzHub. If you install it from npm, you must first install Node.js, as described in the following section.

Software required for downloading the CLI from npm

Node Package Manager, or npm, is a public software repository. The MobileFirst Platform CLI is hosted on JazzHub and npm.

You must install Node.js to be able to download the CLI from npm. For information about installing Node.js, see the Node js web site.

Software required for adapter development

You can create, build, and deploy adapters with the CLI. If you plan to develop adapters, you also need to download and install Maven and put the Maven executable in your system path. For instructions for installing Maven, see Installing Apache Maven.

For more information about MobileFirst adapters, see the Overview of adapters.

Installing the MobileFirst Platform CLI

Install the IBM MobileFirst Platform Command Line Interface (CLI) so that you can use the CLI to develop your app.

Before you begin

- You must have computer with a supported OS X, Windows, or Linux operating system installed. See "System requirements" on page 2-7 for more information about supported operating systems.
- You must have node.js version 4.0.0 or later installed. If you do not have it installed, you can install it as part of the procedure on this page.
- If you want to install the CLI from the MobileFirst Operations Console, you must have access to the MobileFirst Operations Console on an existing MobileFirst Server. This can be a server running locally, such as the MobileFirst Development Server, or it can be another (typically remote) MobileFirst Server.
- If you want to install the CLI directly from npm or JazzHub, you must have internet access.

Note:

- If you need to set up your development environment on a computer that has no internet access, you can install it offline. See How to set up an offline IBM MobileFirst development environment.
- If you are installing the CLI from the IBM MobileFirst Platform Foundation Developer Kit that is already downloaded, you do not need internet access. To install the CLI from the IBM MobileFirst Platform Foundation Developer Kit, see "Installing the IBM MobileFirst Platform Foundation Developer Kit" on page 7-10.

About this task

You can install the CLI from the MobileFirst Operations Console or directly from npm. The CLI is provided as an npm (Node Package Manager) package, and you use the **`npm install`** command to install it.

Note: You can also download the compressed (.zip) file packages from JazzHub. Click the link that starts with `hub.jazz.net` from the npm page at the following URL: `www.npmjs.com/package/mfpdev-cli` to go directly to the JazzHub web page for the CLI.

Procedure

You can install the CLI in two main ways: from the MobileFirst Operations Console or from npm.

- To install the CLI from theMobileFirst Operations Console:
  1. From the MobileFirst Operations Console Dashboard, click **Get Starter Code**.
  2. From the Downloads page, select the **Tools** tab.
  3. Under Developer CLI, click **Download**.
  4. Save the file `mfpdev-cli.tgz` to your local computer.
  5. (Required only if Node.js is not already installed. If Node.js is already installed, skip this step.) Install Node.js 4.0.0 or later. To download and install Node.js, click the link **Node.js to be installed** in the console. This takes you to the Node.js web site. Follow the download and installation instructions there.

Note: Alternatively, you can reach the Node.js web site by clicking this link: Node.js web site

6. Open a command prompt or terminal window and run the following command:

   ```
   npm install -g --no-optional path_cli_file
   ```

   where *path_cli_file* is the full path and name of the downloaded file, including extension. For example, if the file is in the current working directory:

   ```
   npm install -g --no-optional mfpdev-cli.tgz
   ```

- To install the CLI from npm:
  1. (Required only if Node.js is not already installed. If Node.js is already installed, skip this step.) Install Node.js 4.0.0 or later. To install Node.js, go to the Node.js web site and follow the instructions.
  2. Open a command prompt or terminal window and run the following command:

     ```
     npm install -g --no-optional mfpdev-cli.tgz@8.0
     ```

Results

The MobileFirst Platform CLI is installed.

What to do next

Define which MobileFirst Server you want to be the default, if you do not want to use the factory default of `http://localhost:9080`. For more information, see "Defining the target server of the MobileFirst Platform CLI."

Defining the target server of the MobileFirst Platform CLI:

Before you start developing with MobileFirst Platform CLI, you must define the MobileFirst Server that you use for developing and testing.

About this task

The following procedure applies only when you use MobileFirst Development Server that is installed on another computer, or a MobileFirst Server other than MobileFirst Development Server with its default configuration. If you use MobileFirst Development Server locally on your computer, with its default configuration then this procedure does not apply to you.

If the MobileFirst Platform CLI target server is not configured, it connects to `http://localhost:9080/mfpadmin` with *admin* and *admin* as login credentials.

Procedure

1. Run **`mfpdev help server add`** to get a list of arguments to add a server.
2. Run **`mfpdev server add`** with the arguments that you need.
3. Run **`mfpdev server info`** to verify that the server is added.

Example

To add a default MobileFirst Development Server that runs on the computer `testserver.mydomain`, run the command **`mfpdev server add myserver -url http://testerver.mydomain:9080 -login admin -password admin -contextroot mfpadmin -s`**.

Command-line interface (CLI) summary

The **mfpdev** command-line consists of the following commands.

Table 7-1. MobileFirst Platform CLI summary

Command prefix	Command action	Description
mfpdev app	**register**	Registers your app with a MobileFirst Server.
	config	Enables you to specify the back-end server and runtime to use for your app. In addition, for Cordova apps, enables you to configure several additional aspects such as the default language for system messages and whether to do a checksum security check. Other configuration parameters are included for Cordova apps.
	pull	Retrieves an existing app configuration from the server.
	push	Sends an app's configuration to the server.
	preview	Enables you to preview your Cordova app without requiring an actual device of the target platform type. You can view the preview in either the Mobile Browser Simulator or your web browser.
	webupdate	Packages the application resources contained in the `www` directory into a `.zip` file that can be used for the Direct Update process.
mfpdev server	**info**	Displays information about the MobileFirst Server.
	add	Adds a new server definition to your environment
	edit	Enables you to edit a server definition.
	remove	Removes a server definition from your environment.
	console	Opens the MobileFirst Operations Console.
	clean	Unregisters apps and removes adapters from the MobileFirst Server.
mfpdev adapter	**create**	Creates an adapter.
	build	Builds an adapter.
	build all	Finds and builds all of the adapters in the current directory and in its subdirectories.
	deploy	Deploys an adapter to the MobileFirst Server.
	deploy all	Finds all of the adapters in the current directory and in its subdirectories, and deploys them to the MobileFirst Server.
	call	Calls an adapter's procedure on the MobileFirst Server.
	push	Sends an adapter configuration to the server.
	pull	Retrieves an existing adapter configuration from the server.

Table 7-1. MobileFirst Platform CLI summary (continued)

Command prefix	Command action	Description
`mfpdev`	`config`	Sets your configuration preferences for preview browser type, preview timeout value, and server timeout value for the mfpdev command-line interface.
	`info`	Displays information about your environment, including operating system, memory consumption, node version, and command-line interface version. If the current directory is a Cordova application, information provided by the Cordova **`cordova info`**command is also displayed.
	`-v`	Displays the version number of the MobileFirst Platform CLIcurrently in use.
	`-d, --debug`	Debug mode: Produces debug output.
	`-dd. --ddebug`	Verbose debug mode: Produces verbose debug output.
	`-no-color`	Suppresses use of color in command output.
`mfpdev help`	`[<command type > | [<command action>]]`	Displays help for MobileFirst Platform CLI (**`mfpdev`**) commands. With a arguments, displays more specific help text for each command type or command. For more information see "Command-line interface (CLI) help."

Command-line interface (CLI) help

You can run the **`help`** command to display full information about all MobileFirst Platform CLI commands.

The **`mfpdev help`** command displays information about all commands in the CLI. You can display different levels of information as follows:

Table 7-2. Displaying help for MobileFirst Platform CLI commands

To display...	Type in your command window...
A summary of all commands	`mfpdev help`
A summary of all application-related commands	`mfpdev help app`
A summary of all server-related commands	`mfpdev help server`
A summary of all adapter-related commands	`mfpdev help adapter`
Details of a specific command	**`mfpdev help`** *`<command>`* where *`<command>`* is either two or three parts. See Examples.

Examples

`mfpdev help`
: Displays summary of all commands.

`mfpdev help server`
: Displays summary of all server-related commands.

`mfpdev help adapter`
: Displays summary of all adapter-related commands.

`mfpdev help app register`
Displays full description and syntax of **`mfpdev app register`** command.

`mfpdev help config`
Displays full description and syntax of **`mfpdev config`** command.

Interactive mode and direct mode

The IBM MobileFirst Platform Command Line Interface (CLI) supports two modes of operation: interactive mode and direct mode.

The **`mfpdev`** commands have two modes: interactive mode and direct mode. In interactive mode, you enter the command without options, and you are prompted for responses. In direct mode, you enter the full command, including options, and prompts are not provided. When applicable, the prompts are context-sensitive to the target platform of the app, as determined by the directory from which you run the command. Use the up and down arrow keys on your keyboard to move through the selections, and press the **Enter** key when the selection you want is highlighted and preceded by a > character.

Examples

Direct mode
You enter the full command, including its options and arguments on one line and press the **Enter** key.

For example:

```
$ mfpdev server add Server1 --url https://acme.appserver.com:9080 --setdefault --login admin --password abcd999
```

This command creates a server profile for a MobileFirst Server that can be reached at the specified URL, and has the administrative login user name of admin and the password `abcd999`. The name of the new profile is Server1, and it is specified to be the default server profile,

Interactive mode
You enter the command with no options or arguments and press the **Enter** key. You are then prompted to set the available parameters one by one.

Example 1:

```
$ mfpdev server add
```

Entering this command initiates display of the following prompts, in sequence:

```
Enter the name of the new server definition:
Enter the MobileFirst Server administrator login ID: (admin)
Enter the MobileFirst Server administrator password:
Save the admin password for this server? (Y/n)
Enter the context root of the  MobileFirst Server administration services: (mfpadmin)
Enter the MobileFirst Server connection timeout in seconds: (30)
Make this server the default? (Y/n)
```

- Defaults are displayed in parenthesis. Press the **Enter** key to select the default.
- Questions that require a yes or no answer display `(Y/n)` or `(y/N)`. The character in uppercase is the default. Press the **Enter** key to select the default, or `y` (for yes) or `n` (for no) followed by the **Enter** key. For example, if you press **Enter** after the last prompt in the example, the server profile that you are defining becomes the default server profile.

Example 2:

```
$ mfpdev app config
```

Entering this command displays a list of possible configuration keys that are applicable to your app. For example:

```
Select key
Server
Runtime
Direct Update Authenticity Public Key
Language Preferences
iOS Ignore File Extensions
>Changes completed, exit app config.
```

You toggle among the choices by pressing the up or down arrow keys on your keyboard. You can also use the **J** and **L** keys, for down and up respectively. The prompt (>) indicates the current focus. Press the **Enter** key when the > prompt is next to the choice that you want to select.

Global command-line options

Several options work with any MobileFirst Platform CLI command.

You can use the following options with any **mfpdev** command. These options must be entered in each command; they do not persist.

Option	Description
-v, --version	Prints this utility's version.
-d, --debug	Produces debug log output.
-dd, --ddebug	Produces verbose debug log output.
--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.

Configuring the application from the CLI

You can set values for your application's configurable MobileFirst parameters with the **mfpdev app config** command.

You can specify values for the settings that are listed in the following tables. The settings in Table 1 apply to applications for all supported platforms: Cordova, native iOS, native Android, and native Windows. The settings listed in Table 2 apply to Cordova applications only.

You can specify the required settings with the **mfpdev app config** command to use direct mode. You can also enter only the **mfpdev app config** command to complete the information with prompts. See "Interactive mode and direct mode" on page 7-19 for more information about direct and interactive modes.

Table 7-3. Settings for all supported target platforms

Setting	Description
server	Specifies the server profile to use to update MobileFirst Server information. If a value is not passed then the default server profile is used.
runtime	Specifies the runtime to use on the specified MobileFirst Server. The default is mfp.

Table 7-3. Settings for all supported target platforms (continued)

Setting	Description
`language_preferences`	Specifies the default language to use for client system messages. To have the language default to the locale that is set on the mobile device, enter a space. Other possible values are: • English: `en` • French: `fr` • Spanish: `es` The default is English (`en`).

You can specify the following additional settings for Cordova apps:

- The direct update authentication public key
- Whether or not to enable the web resources checksum test
- What file extensions to ignore during the web resources checksum test

For the web resources checksum settings, each possible target platform (Android, iOS, Windows 8, Windows Phone 8, and Windows 10 UWP) has a platform-specific key for use in **`mfpdev`** direct mode. These keys begin with a string that represents the platform name. For example, `windows10_security_test_web_resources_checksum` is a true or false setting that specifies whether to enable the web resources checksum test for Windows10 UWP.

Table 7-4. Settings for Cordova applications only.

Setting	Description
`direct_update_authenticity_public_key`	Specifies the public key for direct update authentication. The key must be in Base64 format. For more information, see "Implementing secure Direct Update on the client side" on page 7-239.
`ios_security_test_web_resources_checksum`	If set to `true`, enables the test for web resources checksum for iOS Cordova apps. The default is `false`.
`android_security_test_web_resources_checksum`	If set to `true`, enables the test for web resources checksum for Android Cordova apps. The default is `false`.
`windows10_security_test_web_resources_checksum`	If set to `true`, enables the test for web resources checksum for Windows 10 UWP Cordova apps. The default is `false`.
`windows8_security_test_web_resources_checksum`	If set to `true`, enables the test for web resources checksum for Windows 8.1 Cordova apps. The default is `false`.
`windowsphone8_security_test_web_resources_checksum`	If set to `true`, enables the test for web resources checksum for Windows Phone 8.1 Cordova apps. The default is `false`.
`ios_security_ignore_file_extensions`	Specifies what file extensions to ignore during web resources checksum testing for iOS Cordova apps. Separate multiple extensions with commas. For example: `jpg,gif,pdf`
`android_security_ignore_file_extensions`	Specifies what file extensions to ignore during web resources checksum testing for Android Cordova apps. Separate multiple extensions with commas. For example:`jpg, gif,pdf`

Table 7-4. Settings for Cordova applications only (continued).

Setting	Description
`windows10_security_ignore_file_extensions`	Specifies what file extensions to ignore during web resources checksum testing for Windows 10 UWP Cordova apps. Separate multiple extensions with commas. For example: `jpg,gif,pdf`
`windows8_security_ignore_file_extensions`	Specifies what file extensions to ignore during web resources checksum testing for Windows 8.1 Cordova apps. Separate multiple extensions with commas. For example: `jpg,gif,pdf`
`windowsphone8_security_ignore_file_extensions`	Specifies what file extensions to ignore during web resources checksum testing for Windows Phone 8.1 Cordova apps. Separate multiple extensions with commas. For example: `jpg,gif,pdf`

Getting started with the MobileFirst CLI

To get started with the CLI, create a MobileFirst Server profile, configure your app, and register your MobileFirst app to the MobileFirst Server.

Before you begin

This sequence of steps assumes that you already have the following on your local computer:

- The IBM MobileFirst Platform Command Line Interface (CLI). For information about installing the CLI, see "Installing the MobileFirst Platform CLI" on page 7-15.
- A MobileFirst application that is under development, with its files contained in a root directory and subdirectories. The application must already include the IBM MobileFirst Platform Foundation SDK.
- Either a local MobileFirst Server running on your computer or connectivity to a remote, running MobileFirst Server. The server can be a MobileFirst Development Server, the server that is provided with the IBM MobileFirst Platform Foundation Developer Kit. For information about installing and starting the MobileFirst Development Server server, see "Setting up the MobileFirst Development Server" on page 7-12. For information about installing a full MobileFirst Server, see "Installing IBM MobileFirst Platform Server" on page 6-2.

About this task

The numbered steps that follow describe a typical initial task flow from the command line. These steps are followed by a series of commands that demonstrates how you can create and manage adapters, including calling adapter procedures, from the command line. Finally, examples are provided of some optional command-line tasks.

Procedure

1. Create at least one MobileFirst Server profile with the **`mfpdev server add`** command. For more information about this command, run **`mfpdev help server add`**.
2. Optional: Configure your app with the **`mfpdev app config`** command. For more information about this command, run **`mfpdev help app config`**.

3. If you are using a local development server, start the MobileFirst Server. For a remote server, verify with the administrator that the server is running. For instructions on starting the server see "Installing IBM MobileFirst Platform Server" on page 6-2.
4. Verify server access by opening the MobileFirst Operations Console for the MobileFirst Server. Run **`mfpdev server console`**. For more information about this command, run **`mfpdev help server console`**.
5. Change directories into your project. For example, run **`cd YourProject`**.
6. Register your app on the MobileFirst Server by running the **`mfpdev app register`**. For more information about this command, run **`mfpdev help app register`**. Also, see the information about registering your app in the IBM Knowledge Center section for your app type. For example, for an iOS app, see "Registering iOS applications to MobileFirst Server" on page 7-37).
7. Optional: (Cordova applications only) Preview your app by using the **`mfpdev app preview`** command. For more information, run **`mfpdev help app preview`**.

Optional additional steps if your app uses an adapter

8. Create an adapter by running **`mfpdev adapter create`**. For more information about this command, run **`mfpdev help adapter create`**.
9. Build the adapter by changing to the adapter directory and running **`mfpdev adapter build`**. For example:

   ```
   $ cd MyAdapter
   $ mfpdev adapter build
   ```

   For more information about this command, run **`mfpdev help adapter build`**.
10. Deploy the adapter by running **`mfpdev adapter deploy`**. For more information about this command, run **`mfpdev help adapter deploy`**.
11. Call procedures on the deployed adapter by running **`mfpdev adapter call`**. For more information about this command, run **`mfpdev help adapter call`**.

Other optional steps

12. Perform other tasks with the CLI whenever the need arises. For example:
    - For cross-platform (Cordova) apps, you can display a preview of the app in the built-in Mobile Browser Simulator or in your web browser.
    - Also for Cordova apps, you can generate and deploy a compressed (.zip) file of web resources to a MobileFirst Server with the **`mfpdev app webupdate`** command. For more information about this command, run **`mfpdev help app webupdate`**.
    - You can display information about the available servers with the **`mfpdev server info`** command. For more information about this command, run **`mfpdev help server info`**.
    - You can replicate app and adapter configuration settings from one MobileFirst Server to another by using the **`mfpdev app pull`** and **`mfpdev app push`** commands. For more information about these commands, run **`mfpdev help app pull`** and **`mfpdev help app push`** .
    - You can modify an existing server profile with the **`mfpdev server edit`** command. For more information about this command, run **`mfpdev help server edit`**.

Setting up an internal Maven repository for offline development

If you do not have online access to The Central Repository, you can share MobileFirst Maven artifacts in the internal repository of your organization.

Before you begin

Make sure that you have installed Apache Maven.

Procedure

1. Download the MobileFirst Platform Foundation Development Kit Installer.
2. Start MobileFirst Server and in a browser, load the MobileFirst Operations Console from the following URL: `http://<your-server-host:server-port>/mfpconsole`.
3. Click **Download Center**. Under **Tools** > **Adapter Archetypes**, click **Download**. The `mfp-maven-central-artifacts-adapter.zip` archive is downloaded.
4. Add the adapter archetypes and security checks to the internal Maven repository by running the `install.sh` script for Linux and Mac, or the `install.bat` script for Windows.
5. The following JAR files are required by adapter-maven-api. Make sure they are located either in developers' local `.m2` folder, or in the Maven repository of your organization. Download them from The Central Repository.
   - javax.ws.rs:javax.ws.rs-api:2.0
   - javax:javaee-web-api:6.0
   - org.apache.httpcomponents:httpclient:4.3.4
   - org.apache.httpcomponents:httpcore:4.3.2
   - commons-logging:commons-logging:1.1.3
   - javax.xml:jaxp-api:1.4.2
   - org.mozilla:rhino:1.7.7
   - io.swagger:swagger-annotations:1.5.6
   - com.ibm.websphere.appserver.api:com.ibm.websphere.appserver.api.json:1.0
   - javax.servlet:javax.servlet-api:3.0.1

Developing the client side of a MobileFirst application

This collection of topics relates to various aspects of developing the client side of a MobileFirst application.

Developing MobileFirst applications

The process for developing applications has steps that are common to all environments: setting up a server, creating an initial server registration and corresponding configuration files, creating a new (or opening an existing) project in your chosen IDE, and adding the necessary SDK files to your IDE project.

Setting up the development environment

Before you can build and run your app, register it with a running MobileFirst Development Server. Read the following topics to learn how to set up your development server, add the MobileFirst libraries to your application, and register it with a server:

- Set up the MobileFirst Development Server, if necessary. See “Setting up the MobileFirst Development Server” on page 7-12.
- Add the MobileFirst libraries for your application.
  - “Developing native applications for iOS in Xcode” on page 7-27
  - “Developing native applications in Android Studio” on page 7-52

  - "Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal" on page 7-65
  - "Developing web applications" on page 7-73
  - "Developing Cordova applications" on page 7-83
- Register the application to the MobileFirst Development Server.
  - "Registering iOS applications to MobileFirst Server" on page 7-37
  - "Registering Android applications to MobileFirst Server" on page 7-59
  - "Registering Windows applications to MobileFirst Server" on page 7-68
  - "Registering web applications to MobileFirst Server" on page 7-80
  - "Registering Cordova applications to MobileFirst Server" on page 7-106

Developing your app

Once you have these resources in your IDE environment, you develop according to the guidelines for your target language and specific behavior of the MobileFirst API for that device platform. In addition, samples and documentation are provided for development in Xcode for iOS, Android Studio for Android, and Visual Studio for Windows.

Getting started with a sample MobileFirst application

Get a copy of a sample MobileFirst application to use as a starting point for developing your own custom application.

Procedure

1. From the navigation sidebar of the IBM MobileFirst Platform Operations Console, select **Download Center**.
2. On the Download Center page, select the **Samples** tab.
3. In the **Application Samples** section of the Samples tab are download options that correspond to samples for the supported development platforms. Select the appropriate download option, and follow the instructions in the dialog window to save the archive file of the sample application to your preferred location. When used with the MobileFirst Development Server from the IBM MobileFirst Platform Foundation Developer Kit, the console displays both local ( ) and remote ( ) download options. When used with another MobileFirst Server, the console displays only remote download options. The local-download options do not require an internet connection.
4. Extract the sample files from the downloaded archive file.

Results

The sample applications contain useful code examples that demonstrate how to use the respective MobileFirst SDK and test the connection to the server. See the `README.md` file in the extracted sample directory for specific information and usage instructions.

You can add the sample to your IDE and use it as the starting point for your development, or use the sample as a general reference for a functioning application.

Acquiring the MobileFirst SDK from the MobileFirst Operations Console

Get a copy of the MobileFirst SDK for manual integration with your application.

Before you begin

Run the IBM MobileFirst Platform Operations Console on the MobileFirst Development Server, which is installed with the IBM MobileFirst Platform Foundation Developer Kit. For more information, see IBM MobileFirst Platform Foundation Developer Kit.

About this task

You can add MobileFirst functionality to an existing application, or upgrade from an earlier version of the MobileFirst SDK, by manually adding the SDK files to your application. Follow the outlined procedure to obtain the required SDK files from the MobileFirst Operations Console.

Note: The SDK files are available locally from the console when using the MobileFirst Development Server, and can be obtained without an internet connection.

Procedure

1. From the navigation sidebar of the MobileFirst Operations Console, select **Download Center**.
2. On the Download Center page, select the **SDKs** tab.
3. From the **SDKs** tab, select the local-download option for the SDK that you want to download:

   Follow the instructions in the dialog window to save the SDK archive file to your preferred location.
4. Extract the SDK files from the downloaded archive file.

Results

For instructions on how to use the SDK, see the `README.md` file in the extracted SDK directory, as well as the following documentation:

- For the iOS SDK, see "Setting up the Xcode project for iOS manually" on page 7-27.
- For the Android SDK, see "Setting up Android Studio projects with Gradle" on page 7-53.
- For the Windows Universal SDK, see "Adding the MobileFirst SDK manually" on page 7-65.
- For the web SDK, see "Adding the MobileFirst SDK to web applications" on page 7-76.
- For the Cordova SDK, see "Adding MobileFirst features to an existing Cordova app" on page 7-91.

Developing native applications for iOS in Xcode

To create an IBM MobileFirst Platform Foundation app, you must set up the IBM MobileFirst Platform Foundation SDK and configuration files, and add them to your iOS Xcode project. Then you can start to develop your app.

Note: MobileFirst development is supported in Xcode from version 7.1 by using iOS 8.0 and later.

To develop a native iOS application, you must add the MobileFirst framework files to your Xcode project and register the app on the IBM MobileFirst Platform Server. See "Registering iOS applications to MobileFirst Server" on page 7-37.

Note: Configuration and setup guidelines are provided for both Swift and Objective-C Xcode projects. You can find many Swift code examples in the Developer Center Tutorials.

You can add the MobileFirst frameworks to your Xcode project in one of the following ways:

- Obtaining a set of MobileFirst framework files and manually copying them to your Xcode project. See "Setting up the Xcode project for iOS manually."
- Using CocoaPods. See "Adding MobileFirst SDK to an iOS Xcode project using CocoaPods" on page 7-29.

The following topics show you how to set up an initial project and start developing.

Methods of setting up your environment

You can prepare your environment for developing MobileFirst applications in either of two ways: by copying the MobileFirst frameworks to your Xcode project or by installing the files using CocoaPods.

After you have set up your environment, you can start developing your iOS code.

Setting up the Xcode project for iOS manually:

You can add MobileFirst functionality to your existing or new Xcode project. The required framework and library files can be generated by IBM MobileFirst Platform Foundation by using IBM MobileFirst Platform Operations Console and added to your Xcode project. The Xcode project must be then configured correctly according to your development goals.

Before you begin

You must have:

- Xcode 7.1 with iOS 8 or higher to develop your code
- A set of MobileFirst SDK files. See "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26.

About this task

Procedure

1. In your Xcode project, add the MobileFirst framework files to your project.
   a. Select the project root icon in the project explorer.

b. Select **File** > **Add Files** and navigate to the folder that contains the framework files created by IBM MobileFirst Platform Operations Console.
c. Click the **Options** button.
d. Select **Copy items if needed** and **Create groups for any added folders**.

   Note: If you do not select the **Copy items if needed** option, the framework files are not copied but are linked from their original location.
e. Select the main project (first option) and select the app target.
f. In the **General** tab, remove any frameworks that would get added automatically to **Linked Frameworks and Libraries**.
g. Required: In **Embedded Binaries**, add the following frameworks:
   - `IBMMobileFirstPlatformFoundation.framework`
   - `IBMMobileFirstPlatformFoundationOpenSSLUtils.framework`
   - `IBMMobileFirstPlatformFoundationWatchOS.framework`
   - `Localizations.bundle`

   Performing this step would automatically add these frameworks to **Linked Frameworks and Libraries**.
h. In **Linked Frameworks and Libraries**, add the following frameworks:
   - `IBMMobileFirstPlatformFoundationJSONStore.framework`
   - `sqlcipher.framework`
   - `openssl.framework`
i. Similarly, add optional frameworks. For more information about available frameworks, see Adding optional frameworks manually.

   Note: These steps copy the relevant MobileFirst frameworks to your project and link them within the **Link Binary with Libraries** list in the **Build Phases** tab. If you link the files to their original location (without choosing the **Copy items if needed** option as described previously), you need to set the **Framework Search Paths** as described below.

2. The frameworks added in Step 1, would be automatically added to the **Link Binary with Libraries** section, in the **Build Phases** tab.
3. Optional: If you did not copy the framework files into your project as described previously , perform the following steps by using the **Copy items if needed** option, in the **Build Phases** tab.
   a. Open the **Build Settings** page.
   b. Find the **Search Paths** section.
   c. Add the path of the folder that contains the frameworks to the **Framework Search Paths** folder.
4. In the **Deployment** section of the **Build Settings** tab, select a value for the **iOS Deployment Target** field that is greater than or equal to 8.0.
5. Optional: From Xcode 7, bitcode is set as the default. For limitations and requirements see "Working with bitcode in iOS apps" on page 7-48. To disable bitcode:
   a. Open the **Build Options** section.
   b. Set **Enable Bitcode** to `No`.
6. Beginning with Xcode 7, TLS must be enforced.

   See "Enforcing TLS-secure connections in iOS apps" on page 7-46.

Results

Your Xcode project is now ready for development.

You must import the headers for the `IBMMobileFirstPlatformFoundation` framework:

Objective C

```
#import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
```

Swift

```
import IBMMobileFirstPlatformFoundation
```

What to do next

Before you can access server resources, you must register your app. See "Registering iOS applications from the MobileFirst Platform CLI" on page 7-37. For details about the `mfpclient.plist` file see "iOS client properties file" on page 7-41.

Adding MobileFirst SDK to an iOS Xcode project using CocoaPods:

You can add MobileFirst functionality to your existing or new Xcode project with CocoaPods. Once your project is set up, you can continue developing your code.

Before you begin

Note: MobileFirst development is supported in Xcode from version 7.1 by using iOS 8.0 and later.

You must have

- CocoaPods installed in your development environment (for more information, see the "Getting Started" guide for CocoaPods installation)
- Xcode 7.1 with iOS 8.0 or higher for your development environment

About this task

You can create a IBM MobileFirst Platform Foundation Xcode project by adding core frameworks and libraries with CocoaPods. Addtional optional CocoaPods may be added.

Procedure

1. Create an Xcode project if one does not exist.
2. Open a command line in the Xcode project folder.
3. Run the **`pod init`** command to create a `Podfile` file.
4. Open the new `Podfile` file also at the Xcode project root.
5. Comment out or remove the entire existing content.
6. Add the following lines including the iOS version and save the changes:

```
source 'https://github.com/CocoaPods/Specs.git'
use_frameworks!
platform :ios, 9.0
target :name-of-the-target-in-xcode-project do
  pod 'IBMMobileFirstPlatformFoundation'
end
```

7. Optional: If you want to add any optional features, add additional pods. For a list of available features see Table 7-7 on page 7-34. Add an additional line for each pod, for example:

   ```
   pod 'IBMMobileFirstPlatformFoundationJSONStore'
   ```

   Note:

   The previous syntax imports the latest version of the `IBMMobileFirstPlatformFoundation` pod. If you are not using the latest version of MobileFirst, you need to add the full version number, including the major, minor, and patch numbers. The patch number is in the format *YYYYMMDDHH*. For example, for importing the specific patch version 8.0.2016021411 of the `IBMMobileFirstPlatformFoundation` pod the line would look like this:

   ```
   pod 'IBMMobileFirstPlatformFoundation', '8.0.2016021411'
   ```

   Or to get the last patch for the minor version number the syntax such is

   ```
   pod 'IBMMobileFirstPlatformFoundation', '~>8.0.0'
   ```

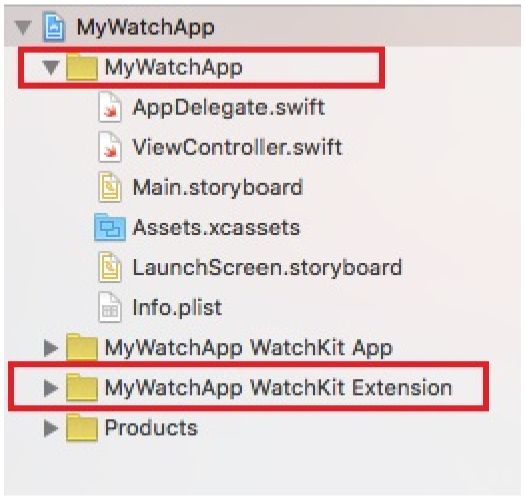
8. Optional: If you are developing for watchOS your `Podfile` must contain sections corresponding to the main app and the watchOS extension:

```
#use the name of the app
xcodeproj 'MyWatchApp'

use_frameworks!

#use the name of the iOS target
target :MyWatchApp do
    platform :ios, 9.0
    pod 'IBMMobileFirstPlatformFoundation'
    end
#use the name of the watch extension target
target :MyWatchApp WatchKit Extension do
    platform :watchos, 2.0
    pod 'IBMMobileFirstPlatformFoundation'
end
```

The previous targets must match your main iOS app and watchOS extension:



The main app section can contain any of the frameworks documented here: Table 7-7 on page 7-34. However, only the `IBMMobileFirstPlatformFoundation` pod is supported for the watchOS extension. For more information on watchOS, see "Developing for watchOS 2" on page 7-48.

9. Verify that the Xcode project is closed.
10. Run the **`pod install`** command. This command installs the `IBMMobileFirstPlatformFoundation` pod and any other pods that are specified in the `Podfile` and their dependencies. It then generates the pods project, and integrates the client project with the MobileFirst SDK. It also adds other required dependencies.
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. The main framework is imported like this:

    Swift:

    ```
    import IBMMobileFirstPlatformFoundation
    ```

    Objective C

    ```
    #import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
    ```
13. Beginning with iOS9 TLS must be enforced, see "Enforcing TLS-secure connections in iOS apps" on page 7-46.

Results

You can now start developing your native iOS application with the IBM MobileFirst Platform Foundation integration.

What to do next

Before you can access server resources, you must register your app. See "Registering iOS applications from the MobileFirst Platform CLI" on page 7-37. For details about the `mfpclient.plist` file see "iOS client properties file" on page 7-41

Adding optional iOS frameworks

MobileFirst iOS functionality is provided by a collection of frameworks that can be added to your app. Only one of these frameworks is required (`IBMMobileFirstPlatformFoundation`). You can add optional frameworks according to the features you want to implement in your app. You reduce the size of the app by including only the frameworks required by your chosen features.

For an existing MobileFirst Xcode project , you can add frameworks manually by linking the frameworks in Xcode or by using Cocoapods.

Table 7-5. Optional frameworks for iOS

Feature	**Frameworks (linked in the Link Binary with Libraries list in the Build Phases tab)**

JSONStore	`IBMMobileFirstPlatformFoundationJSONStore`
	
	`SQLCipher`
	
	In addition, import the `IBMMobileFirstPlatformFoundationJSONStore` header to your code. For more information on setup, see "JSONStore" on page 7-134.

OpenSSL	`openssl`
	
	`IBMMobileFirstPlatformFoundationOpenSSLUtils`
	
	For more information on OpenSSL, see "Enabling OpenSSL for iOS" on page 7-47

Table 7-5. Optional frameworks for iOS (continued)

Feature	Frameworks (linked in the Link Binary with Libraries list in the Build Phases tab)
Push	`IBMMobileFirstPlatformFoundationPush` For more information, see "Push notification" on page 7-248. In addition, import the `IBMMobileFirstPlatformFoundationPush` header to your code.
watchOS	`IBMMobileFirstPlatformFoundationWatchOS`. The watchOS framework requires a different structure for the Xcode project. For information on adding the watchOS framework, see Adding watchOS frameworks.

Adding optional frameworks manually:

You can add optional MobileFirst features to your existing MobileFirst app project. The required framework and library files can be generated by IBM MobileFirst Platform Foundation by using IBM MobileFirst Platform Operations Console and added to your Xcode project. The Xcode project must be then configured correctly according to your development goals.

Before you begin

You must have

- A set of MobileFirst framework files (see "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26).
- An existing Xcode MobileFirst project which contains the core frameworks and libraries (see "Setting up the Xcode project for iOS manually" on page 7-27 or "Adding MobileFirst SDK to an iOS Xcode project using CocoaPods" on page 7-29).

About this task

Optional frameworks

In addition to the core MobileFirst framework many optional frameworks are available. You can limit the size of your app by including only those frameworks required by the features you use. Some optional frameworks require imported headers in your code.

Table 7-6. Optional frameworks for iOS

Feature	Frameworks (linked in the Link Binary with Libraries list in the Build Phases tab)

JSONStore	`IBMMobileFirstPlatformFoundationJSONStore`
	
	`SQLCipher`
	
	In addition, import the `IBMMobileFirstPlatformFoundationJSONStore` header to your code. For more information on setup, see "JSONStore" on page 7-134.

Table 7-6. Optional frameworks for iOS (continued)

Feature	Frameworks (linked in the Link Binary with Libraries list in the Build Phases tab)

OpenSSL	`openssl`
	
	`IBMMobileFirstPlatformFoundationOpenSSLUtils`
	
	For more information on OpenSSL, see "Enabling OpenSSL for iOS" on page 7-47

Push	`IBMMobileFirstPlatformFoundationPush` For more information, see "Push notification" on page 7-248. In addition, import the `IBMMobileFirstPlatformFoundationPush` header to your code.
watchOS	`IBMMobileFirstPlatformFoundationWatchOS`. The watchOS framework requires a different structure for the Xcode project. For information on adding the watchOS framework, see Adding watchOS frameworks.

Procedure

1. In your Xcode project, add the MobileFirst framework files to your project.
   a. Select the project root icon in the project explorer.
   b. From the **File** menu, choose the **Add Files** option and navigate to the folder that contains the framework files.
   c. Click the **Options** button.
   d. Select **Copy items if needed** and **Create groups for any added folders** options.

      Note: If you do not select the **Copy items if needed** option, the framework files are not copied but are linked from their original location.
   e. Select the main project (first option) in the **Add to targets** box.
   f. Choose the framework files (from the previous table) relevant to your project according to your chosen features.
   g. Click **Add**.

      Note: These steps copy the relevant MobileFirst frameworks to your project and link them within the **Link Binary with Libraries** list in the **Build Phases** tab. If you link the files to their original location (without choosing the **Copy items if needed** option as described previously) you need to set the **Framework Search Paths** as described below.
2. Optional: If you did not copy the framework files into your project as described previously, using the **Copy items if needed** option, in the **Build Phases** tab:
   a. Open the **Build Settings** page.
   b. Find the **Search Paths** section.
   c. Add the path of the folder that contains the frameworks to the **Framework Search Paths** folder.

Results

You now have additional frameworks added to your project. Add the required headers to your code according to the Table 7-6 on page 7-32 table.

You must import the headers for some of the frameworks. The syntax depends on the development language:

Objective C:

```
#import <IBMMobileFirstPlatformFoundation/[frameworkname].h>
```

Swift:

```
import [frameworkname]
```

What to do next

Before you can accesss server resources, you must register your app. See “Registering iOS applications from the MobileFirst Platform CLI” on page 7-37

Adding optional frameworks with CocoaPods:

You can add MobileFirst functionality to your existing MobileFirst Xcode project with CocoaPods.

Before you begin

You must have CocoaPods installed in your development environment. In addition you must have a fully functional MobileFirst Xcode project that contains the core MobileFirst framework and libraries. For more information, see “Setting up the Xcode project for iOS manually” on page 7-27 or “Adding MobileFirst SDK to an iOS Xcode project using CocoaPods” on page 7-29.

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. The SDK contains the following optional pods for MobileFirst development:

Table 7-7. Pods for installing optional frameworks

Pod	**Feature**
`IBMMobileFirstPlatformFoundationPush`	Adds the `IBMMobileFirstPlatformFoundationPush` framework for enabling Push. For more information, see “Push notification” on page 7-248.
`IBMMobileFirstPlatformFoundationJSONStore`	Implements the JSONStore feature. Include this pod in your Podfile if you intend to use the JSONStore feature in your app. See “JSONStore” on page 7-134.
`IBMMobileFirstPlatformFoundationOpenSSLUtils`	Contains the MobileFirst embedded OpenSSL feature and loads automatically the `openssl` framework. Include this pod in your Podfile if you intend to use the OpenSSL provided by MobileFirst. For more information on OpenSSL options, see “Enabling OpenSSL for iOS” on page 7-47.

Procedure

1. Open a command line terminal at the location of your Xcode project.
2. Run the **`pod init`** command to create a `Podfile` file.
3. Open the new `Podfile` file also at the Xcode project root.
4. Comment out or remove the entire existing content.
5. Add the following lines including the iOS version and save the changes:

   ```
   use_frameworks!
   platform :ios, [version]
   pod '[pod_name]'
   ```

   For example for the OpenSSL pod using Xcode 9 the file would look like this:

   ```
   use_frameworks!
   platform :ios, 9.0
   pod 'IBMMobileFirstPlatformFoundationOpenSSLUtils'
   ```

   Note: The previous syntax imports the latest version of the `IBMMobileFirstPlatformFoundationOpenSSLUtils` pod. If you are are not using the latest version of MobileFirst, you need to indicate the version. For example, for importing the specific patch version 8.0.2016021411 for `IBMMobileFirstPlatformFoundationOpenSSLUtils` the line would look like this:

   ```
   pod 'IBMMobileFirstPlatformFoundationOpenSSLUtils', '8.0.2016021411'
   ```
6. Optional: If you are developing for watchOS your `Podfile` must contain sections corresponding to the main app and the watchOS extension:

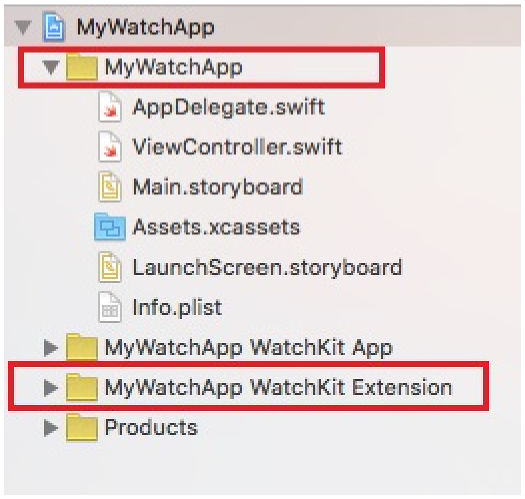
```
#use the name of the app
xcodeproj 'MyWatchApp'

use_frameworks!

#use the name of the iOS target
target :MyWatchApp do
    platform :ios, 9.0
    pod 'IBMMobileFirstPlatformFoundation'
    end
#use the name of the watch extension target
target :MyWatchApp WatchKit Extension' do
    platform :watchos, 2.0
    pod 'IBMMobileFirstPlatformFoundation'
end
```

Note: See the previous note about pod versions.

The targets must match your main iOS app and watchOS extension:



The main app section can contain any of the frameworks documented here. However, only the `IBMMobileFirstPlatformFoundation` pod is supported for the watchOS extension. For more information on watchOS, see "Developing for watchOS 2" on page 7-48.

7. Verify that the Xcode project is closed.
8. Run the **`pod install`** command. This command installs the `IBMMobileFirstPlatformFoundation` pod and any other pods that are specified in the `Podfile` and their dependencies. It then generates the pods project, and integrates the client project with the MobileFirst SDK. It also adds the required non-MobileFirst dependencies.
9. 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.
10. You will need to import the headers for some of the frameworks. The main framework is imported like this: If you are using Push or JSONStore you need to include an independent import:

    Push For Objective C:

    ```
    #import
    <IBMMobileFirstPlatformFoundationPush/IBMMobileFirstPlatformFoundationPush.h>
    ```

    For Swift:

    ```
    import IBMMobileFirstPlatformFoundationPush
    ```

    JSONStore

    For Objective C:

    ```
    #import <IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJSONStore.h>
    ```

    For Swift:

    ```
    import IBMMobileFirstPlatformFoundationJSONStore
    ```

    watchOS

    For Objective C:

    ```
    #import <IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationWatchOS.h>
    ```

    For Swift:

    ```
    import IBMMobileFirstPlatformFoundationWatchOS
    ```

Results

Your Xcode project can now include optional IBM MobileFirst Platform Foundation features.

What to do next

Before you can accesss server resources, you must register your app. See "Registering iOS applications to MobileFirst Server"

Registering iOS applications to MobileFirst Server

Register your iOS application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app. You can register your app by using the IBM MobileFirst Platform Command Line Interface (CLI) or the IBM MobileFirst Platform Operations Console.

Registering iOS applications from the MobileFirst Platform CLI:

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to register your iOS application to an instance MobileFirst Server.

Before you begin

- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have an instance of MobileFirst Server running. The server can be running locally, or it can be a remote server, but it must be reachable from your local computer. For more information, see "Setting up the MobileFirst Development Server" on page 7-12, "Installing the IBM MobileFirst Platform Foundation Developer Kit" on page 7-10, or "Installing IBM MobileFirst Platform Server" on page 6-2.
- You must have a native iOS application on your local computer.

About this task

Once you have the client side of your iOS application initially defined, you can prepare for further development tasks by registering it to a MobileFirst Server.

Tip: If the **`mfpdev app register`** command cannot determine the bundle ID, it will prompt you to enter it. This can occur if value of the CFBundleIdentifier key in the `Info.plist` file contains variables. To avoid this, manually edit the`Info.plist` file and enter the bundle ID without variables before you run the **`mfpdev app register`** command.

Procedure

1. Check that the target MobileFirst Server is up and running.
2. Navigate to the directory that contains your app, or one of its subdirectories.
3. Register your app to the server. Use one of the following procedures:
   - To register the app to the default server, run the following command:
     ```
     mfpdev app register
     ```

Note: If you have not previously set a default server and a MobileFirst Server is running on the local system, this command registers the app to the local MobileFirst Server, and this server is made the default.

- To register your app to a server that is not the default server:
  a. Create a server profile by running the **mfpdev server add** command. For example:
  ```
  mfpdev server add Server1 -url https://company.mobile.com:9080 -login admin -password secre
  ```
  For more information about the **mfpdev server add command**, run `mfpdev help server add`.
  b. To register your app to the server that you just defined, run the **mfpdev app register** command, and specify the server profile that you just created. For example:
  ```
  mfpdev app register Server1
  ```

For more information about this command, including optional parameters, run **mfpdev help app register**.

Results

The app is registered to the target server. Data about the app that is obtained from the platform properties file (`Info.plist`) such as application name, version number, and app ID is sent to the server. If the client properties file `mfpclient.plist` already exists, it is updated with the value of the server's URL. If the file did not exist, a `mfpclient.plist` file that includes the server's URL is created at the root of your project.

Note: If an existing `mfpclient.plist` file resides in another directory in the project besides the root directory, the file will be updated with the server's URL.

What to do next

Copy the `mfpclient.plist` file and register it in your Xcode project. For more information, see "iOS client properties file" on page 7-41.

You can proceed with other development tasks that depend on the MobileFirst Server. For example, you can preview your app, test your app's security features, and manage your app from the MobileFirst Operations Console.

Registering iOS applications from the MobileFirst Operations Console:

Register your iOS application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app.

Before you begin

You must have the IBM MobileFirst Platform Operations Console running on the MobileFirst Server targeted for registration. For more information, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.

About this task

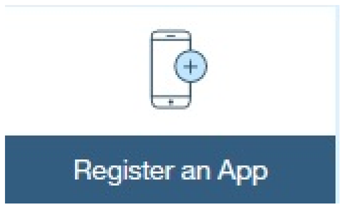
You can register an app on the server before or after setting up the Xcode environment see Methods of setting up your environment. You must register your app to the server before developing code that accesses server resources.

Procedure

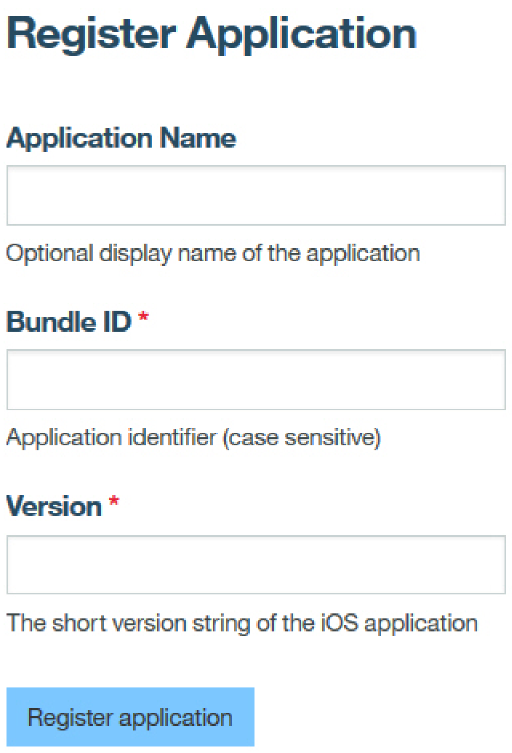
1. In the MobileFirst Operations Console, navigate to the Register Application page by using one of the following methods:
   - In the navigation sidebar, select **New** next to **Appilcations**.

     Applications (0)
New

   - From the **Dashboard**, select **Register an App**.

2. On the **Register an Application** page, fill in the following values for the application you are registering:

   Register Application

   Application Name

   Optional display name of the application

   Bundle ID *

   Application identifier (case sensitive)

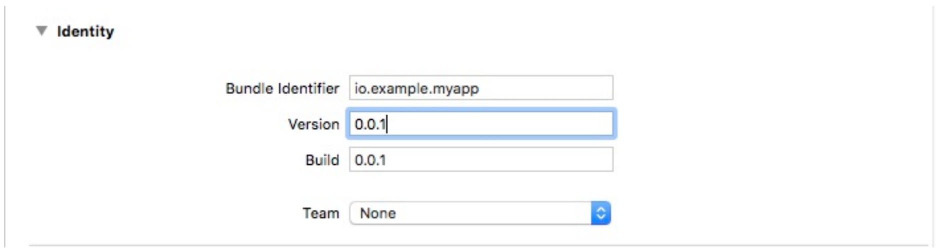
   Version *

   The short version string of the iOS application

   Register application

   - **Application Name**: This is for display and can be any convenient value. It is optional.
   - **Bundle ID**: The bundle identifier from the Xcode project.
   - **Version**: The version number of your iOS app.

     The version number and bundle identifier are reported in the **Identity** section of the **General** tab of your Xcode project.



Identity

Bundle Identifier: io.example.myapp

Version: 0.0.1

Build: 0.0.1

Team: None

These correspond to the values in the `info.plist` file.

Table 7-8. Registration values for iOS projects in the info.plist

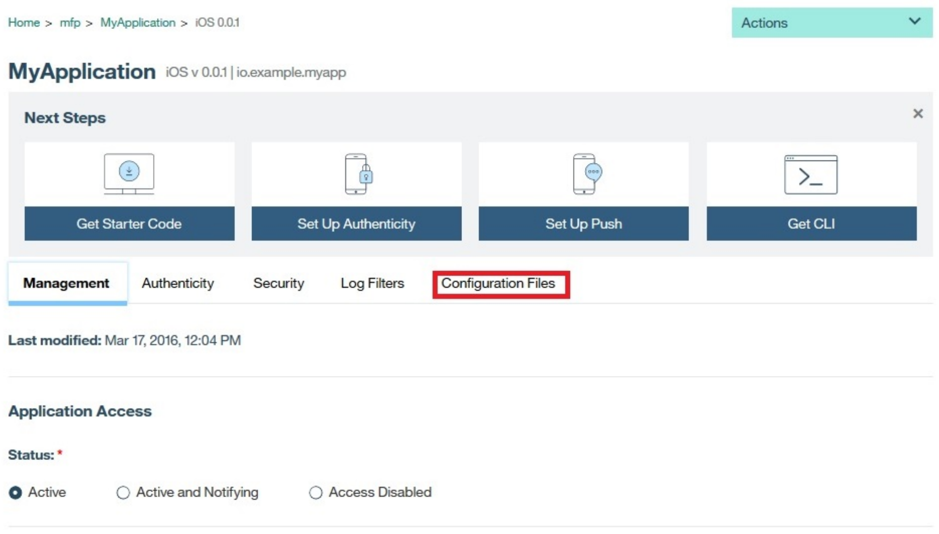
registration value	**parameter in `info.plist`**
Bundle identifier	`CFBundleIdentifier`
Version	`CFBundleShortVersionString`

Note: In the `info.plist` for projects created by Xcode, these values are represented by variables. Therefore you need to check these values in the Xcode project itself.

3. Click the **Register application** button.
4. From the main **Dashboard** page, your application is now listed under the default **mfp** runtime.

Note: The `mfp` runtime is the default value for the `WLServerContext` parameter in the `mfpclient.plist` file below.

5. Click the application name to display the main configuration page for your app.
6. The main page for your app displays different configuration options for the server-side registration of your app.



Last modified: Mar 17, 2016, 12:04 PM

Application Access

Status: *

[● Active] ○ Active and Notifying ○ Access Disabled

Click the **Configuration Files** tab.

7. The **Client Configuration File** tab displays a template for creating your `mfpclient.properties` file. This file is used for connecting the client app to the server.

Results

Your iOS application is registered on the server.

What to do next

To complete the client-server registration, you must complete the required properties in the mfpclient.plist file, copy, and register it in your Xcode project. For more information, see "iOS client properties file."

iOS client properties file

The mfpclient.plist file defines the client-side properties used for registering your iOS app on the MobileFirst server.

The mfpclient.plist client property file contains the necessary information for connecting to the server. The app registration consists of both the server- and client-side components. Before you use the mfpclient.plist file in your native application for iOS, you must define the properties as specified in the following table. This file is created automatically when you register your app with the IBM MobileFirst Platform Command Line Interface (CLI). For more information, see "Registering iOS applications from the MobileFirst Platform CLI" on page 7-37. If you register your application by using the IBM MobileFirst Platform Operations Console, you must in addition create the file manually, reference it within your Xcode project by selecting **File** > **Add Files**.

Table 7-9. Properties of the mfpclient.plist file

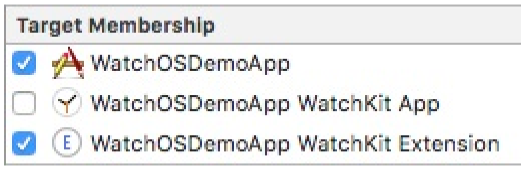
Property	Description	Example values
wlServerProtocol	The communication protocol with the MobileFirst Server.	http or https
wlServerHost	The host name of the MobileFirst Server.	192.168.1.63
wlServerPort	The port of the MobileFirst Server.	9080
wlServerContext	The server context.	/mfp/
wlPlatformVersion	The MobileFirst version	8.0.20160214
languagePreferences	Preferred language.	en

Here is an example mfpclient.plist file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>protocol</key>
 <string>http</string>
 <key>host</key>
 <string>9.148.49.221</string>
 <key>port</key>
 <string>9080</string>
 <key>wlServerContext</key>
 <string>/mfp/</string>
 <key>platformVersion</key>
 <string>8.0.20160309</string>
 <key>languagePreferences</key>
 <string>en</string>
</dict>
</plist>
```

The `mfpclient.plist` file must be added to the root folder of the Xcode project, referenced by the Xcode project, and linked to a target. To link the `mfpclient.plist` file to one or more Xcode targets:

1. In Xcode, from the **File** > **Add File** menu, navigate to the `mfpclient.plist` file,, select the file and click **OK**.
2. After the `mfpclient.plist` file appears in the navigation tree, select it to display the **Target Membership** box. Select the relevant targets.


Target Membership
☑ WatchOSDemoApp
☐ WatchOSDemoApp WatchKit App
☑ WatchOSDemoApp WatchKit Extension

After you register the `mfpclient.plist` file within your Xcode project and register your app on the server, you can start to develop code for accessing server resources.

Creating some initial code in iOS

A simple startup process for iOS is described here with short samples for both Objective C and Swift. Before trying the sample, make sure you have imported the frameworks and added the necessary imports.

Accessing a server resource

In the `ViewController.m` file, after the **ViewController** loads (in the `viewdidLoad` method of the `ViewController`), you can create a resource request without first creating a client.

Objective C

```
((void)viewDidLoad{
    [super viewDidLoad];
    NSURL*url=[NSURL URLWithString:@"/adapters/javaAdapter/users/world"];
    WLResourceRequest*request=[WLResourceRequest requestWithURL:url method:WLHttpMethodGet];
    [request sendWithCompletionHandler:^(WLResponse*response,NSError*error){
        if(error!=nil){
            NSLog(@"Failure: %@",error.description);
        }
        else if(response!=nil){
            // Will print "Hello world" in the Xcode Console.
            NSLog(@"Success: %@",response.responseText);
        }
    }
     ];
}
```

Swift

```
override func viewDidLoad() {
  super.viewDidLoad()
  let url = NSURL(string: "/adapters/javaAdapter/users/world")
  let request = WLResourceRequest(URL: url, method: WLHttpMethodGet)
  request.sendWithCompletionHandler {
   (WLResponse response, NSError error) -> Void in
    if (error != nil){
            NSLog("Failure: " + error.description) }
     else if (response != nil){
            NSLog("Success: " + response.responseText) }
    }
}
```

Testing the server connection without a server resource

If you have not created any server resources and you want to test the server connection, you can test the token access. If you have not created any security checks, the access token should be available. As in the previous example, in the `ViewController.m` file, after the **ViewController** loads, you can request token access without any scope.

Objective C

```
(void)viewDidLoad{
    [super viewDidLoad];
    [[WLAuthorizationManager sharedInstance] obtainAccessTokenForScope: @"" withCompletionHandler:^(AccessToken *accessToken, NSError *error) {
        if (error != nil){
            NSLog(@"Failure: %@",error.description);
        }
        else if (accessToken != nil){
            NSLog(@"Success: %@",accessToken.value);
        }

    }];
}
```

Swift

```
override func viewDidLoad() {
  super.viewDidLoad()
  WLAuthorizationManager.sharedInstance().obtainAccessTokenForScope(nil) { (token, error) -> Void in
            if (error != nil) {
               print("Did not recieve an access token from server: " + error.description)
            } else {
               print("Received access token value: " + token.value)
            }
        }
}
```

For more information about logging, see "Logger SDK" on page 11-37.

For more information about adapters, see "Client access to adapters" on page 7-231.

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 for iOS in Xcode" on page 7-27.
2. In the Xcode IDE, create a Swift file and name it `OCLoggerSwiftExtension.swift`.

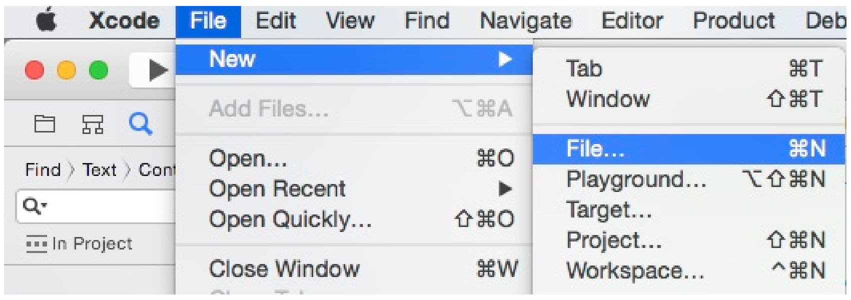
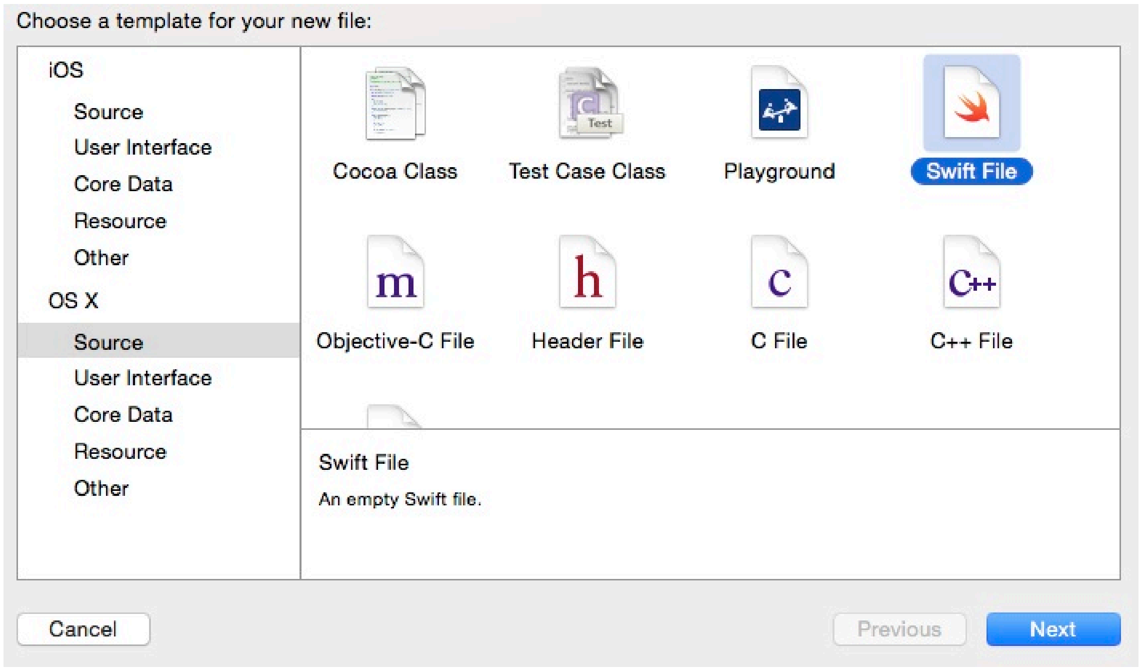


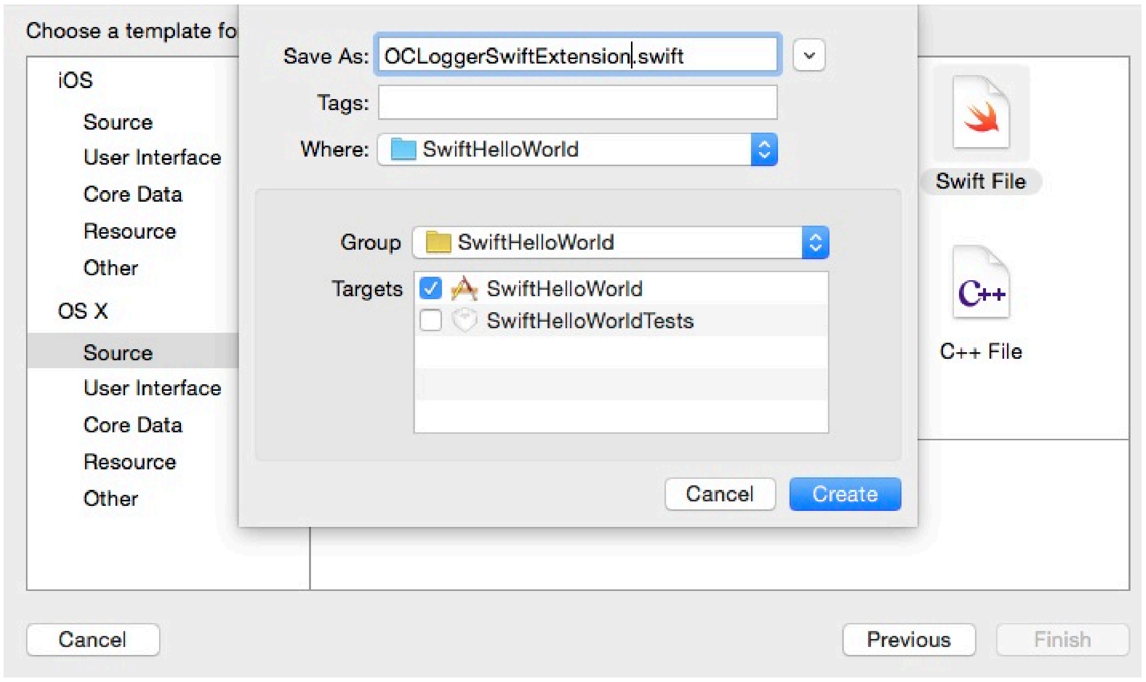
Figure 7-1. New Xcode File



Swift File

An empty Swift file.

Figure 7-2. Swift File Type



Save As: OCLoggerSwiftExtension.swift

Tags:

Where: SwiftHelloWorld

Group: SwiftHelloWorld

Targets: [✓] SwiftHelloWorld, [ ] SwiftHelloWorldTests

Figure 7-3. 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<Stri
  }

  func logDebugWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args), userInfo:Dictionary<S
  }

  func logInfoWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_INFO, message: message, args:getVaList(args), userInfo:Dictionary<St
  }

  func logWarnWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_WARN, message: message, args:getVaList(args), userInfo:Dictionary<St
  }

  func logErrorWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args), userInfo:Dictionary<S
  }

  func logFatalWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args), userInfo:Dictionary<S
  }

  func logAnalyticsWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args), userInfo:Dictiona
  }

  //Log methods with metadata

  func logTraceWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarA
    logWithLevel(OCLogger_TRACE, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logDebugWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarA
    logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logInfoWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarAr
    logWithLevel(OCLogger_INFO, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logWarnWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarAr
    logWithLevel(OCLogger_WARN, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logErrorWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarA
    logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logFatalWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarA
    logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logAnalyticsWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: C
    logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args), userInfo:userInfo
  }
}
```

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 :: I
```

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 protocol and bypass the iOS 9 requirement for development purposes.

About this task

Apple 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.

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>
```

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.

Enabling OpenSSL for iOS

The MobileFirst iOS SDK uses native iOS APIs for cryptography. You can configure the IBM MobileFirst Platform Foundation V8.0.0 to use the OpenSSL cryptography library in iOS apps.

Encryption/decryption is provided with the following APIs: `WLSecurityUtils.encryptText()` and `WLSecurityUtils.decryptWithKey()`

Option 1: Native encryption and decryption

Native encryption and decryption is provided by default, without using OpenSSL. This is equivalent to explicitly setting the encryption or decryption behavior as follows:

```
WLSecurityUtils enableOSNativeEncryption:YES
```

Option 2: Enabling OpenSSL

OpenSSL is disable by default. To enable it, proceed as follows:

1. Install the OpenSSL frameworks:
   - With CocoaPods: Install the `IBMMobileFirstPlatformFoundationOpenSSLUtils` pod with CocoaPods. See Adding OpenSSL with CocoaPods.
   - Manually in Xcode: Link the `IBMMobileFirstPlatformFoundationOpenSSLUtils` and `openssl` frameworks manually in the **Link Binary With Libraries** section of the **Build Phases** tab. See Adding OpenSSL frameworks manually.
2. The following code enables the **OpenSSL option** for the encryption/ decryption:

   `WLSecurityUtils enableOSNativeEncryption:NO`The code will now use the OpenSSL implementation if found and otherwise throw an error if the frameworks are not installed correctly.

With this setup, the encryption/decryption calls use OpenSSL as in previous versions of the product.

Migration options

If you have an MobileFirst project that was written in an earlier version, you might need to incorporate changes to continue using OpenSSL.

- If the application is not using encryption/decryption APIs and no encrypted data is cached on the device, no action is needed.
- If the application is using encryption/decryption APIs, you have the option of using these APIs with or without OpenSSL.

Migrating to native encryption

1. Make sure the default native encryption/decryption option is chosen (see **Option 1**).
2. **Migrating cached data:** If the previous installation of IBM MobileFirst Platform Foundation saved encrypted data to the device using OpenSSL, OpenSSL frameworks must be installed as described in **Option 2**. The first time the application attempts to decrypt the data it will fall back to OpenSSL and then encrypt it using native encryption. If the OpenSSL framework is not installed an error is thrown. This way the data will be auto-migrated to native encryption allowing subsequent releases to work without the OpenSSL framework.

Continuing with OpenSSL

If OpenSSL is required use the setup described in Option 2.

Working with bitcode in iOS apps

Starting with Xcode 7, bitcode is supported and enabled by default for all new projects.

About this task

IBM MobileFirst Platform Foundation supports bitcode with some limitations (see below).

Procedure

1. Go to the **Build Settings** tab for your Xcode project.
2. In the Build Options group, set **Enable Bitcode** to **Yes** or **No**.

   Note: IBM MobileFirst Platform Foundation application-authenticity security check is not supported with bitcode. For more information see "Enabling the application-authenticity security check" on page 7-282.

   Note: Bitcode is required for all watchOS projects.

Developing for watchOS 2

You can develop your watchOS 2 app with MobileFirst by setting the correct frameworks, libraries, and build settings in Xcode.

Setting up watchOS 2 development in Xcode:

To set up the development environment for watchOS 2, create the Xcode project, add the watchOS 2 framework, and set up the necessary targets.

About this task

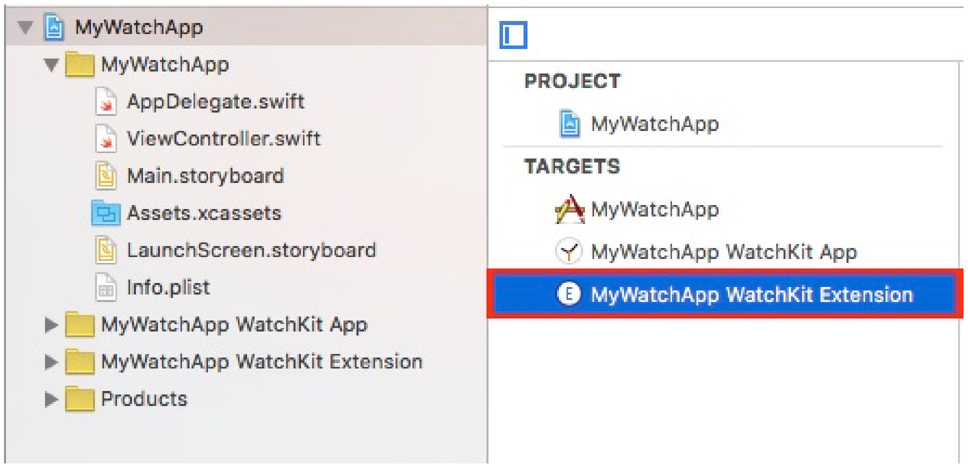
You can create your watchOS 2 project in Xcode and add the watchOS 2 framework manually or with CocoaPods.

Procedure

1. Create a watchOS 2 app in Xcode.

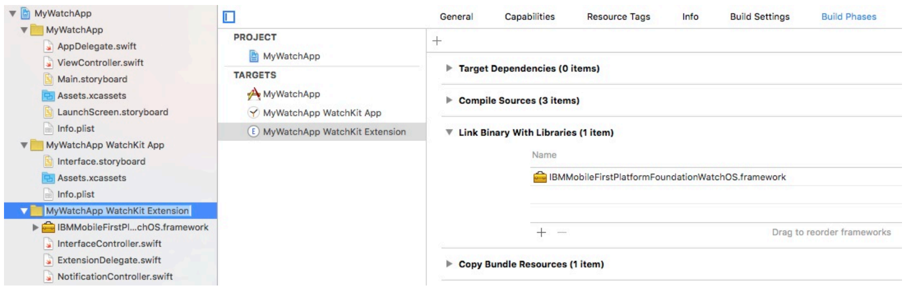
a. Choose the **File->New->Project** option; the **Choose a template for your new project** dialog appears.
b. Choose the **watchOS 2/Application** option, click **Next**.
c. Name the project and click **Next**.
d. From the navigation dialog, choose the project folder.

The project navigation tree now contains a main app folder and a [project name] `WatchKit Extension` folder and target.



2. Add the MobileFirst watchOS 2 framework.
   - To install the necessary frameworks with CocoaPods, see "Adding MobileFirst SDK to an iOS Xcode project using CocoaPods" on page 7-29 and specifically the step for watchOS 2 (Adding frameworks for watchOS).
   - To install the necessary frameworks manually:
     a. Obtain the watchOS 2 framework from MobileFirst Operations Console. See "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26.
     b. Select the [project name] **WatchKit Extension** folder in the left navigation pane.
     c. From the **File** menu, choose **Add Files**.
     d. Click the **Options** button and select the following:
        1) **Copy items if needed** and **Create groups** options.
        2) [project name] **WatchKit Extension** in the **Add to targets** section.
     e. Click **Add**.

     Now when you select the [project name] **WatchKit Extension** in the **Targets** section:
     – The framework path appears in the **Framework Search Paths** setting in the **Search Paths** section of the **Build Settings** tab.
     – The **Link Binary With Libraries** section of the **Build Phases** tab lists the `IBMMobileFirstPlatformFoundationWatchOS.framework` file:

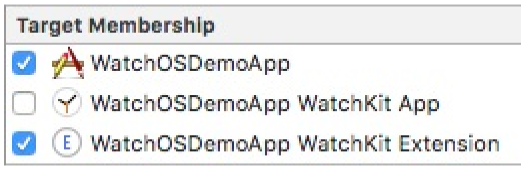

MyWatchApp
MyWatchApp
AppDelegate.swift
ViewController.swift
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
MyWatchApp WatchKit App
Interface.storyboard
Assets.xcassets
Info.plist
MyWatchApp WatchKit Extension
IBMMobileFirstPl...chOS.framework
InterfaceController.swift
ExtensionDelegate.swift
NotificationController.swift
PROJECT
MyWatchApp
TARGETS
MyWatchApp
MyWatchApp WatchKit App
MyWatchApp WatchKit Extension
General
Capabilities
Resource Tags
Info
Build Settings
Build Phases
Target Dependencies (0 items)
Compile Sources (3 items)
Link Binary With Libraries (1 item)
Name
IBMMobileFirstPlatformFoundationWatchOS.framework
Drag to reorder frameworks
Copy Bundle Resources (1 item)

For information on how to set up frameworks for the main iOS app see "Setting up the Xcode project for iOS manually" on page 7-27.

Note: WatchOS 2 requires bitcode. From Xcode 7 the **Build Options** is set to **Enable Bitcode Yes** (**Build Settings** tab, **Build Options** section).

3. Register both the main app and the WatchKit extension on the server.
   a. Run `mfpdev app register` for each Bundle ID: .
      - `com.worklight.[project_name]`
      - `com.worklight.[project_name].watchkitextension`

      This creates two registered apps on the server. For more information on registering iOS apps see "Registering iOS applications to MobileFirst Server" on page 7-37.
   b. In Xcode, from the **File->Add File** menu, navigate to the `mfpclient.plist` file created by `mfpdev` and add it to the project.
   c. Once the `mfpclient.plist` appears in the navigation tree select it to display the **Target Membership** box. Select the **WatchOSDemoApp WatchKit Extension** target in addition to the **WatchOSDemoApp**.



Target Membership

✓ WatchOSDemoApp
☐ WatchOSDemoApp WatchKit App
✓ WatchOSDemoApp WatchKit Extension

4. Beginning with Xcode 7 TLS must be enforced, see "Enforcing TLS-secure connections in iOS apps" on page 7-46. Note that both the main app folder and the WatchKit extension folder have `info.plist` files that need to be updated accordingly.

Results

The Xcode project now contains a main app and a watchOS 2 app, each can be developed independently. For Swift, the entry point for the watchOS 2 app is the `InterfaceController.swift` file in the [project name] **watchKit Extension** folder. For Objective C the entry point is `ViewController.m`.

To use the MobileFirst watchOS 2 API in your code, import the relevant header:

For Objective C:

```
#import <IBMMobileFirstPlatformFoundationWatchOS/IBMMobileFirstPlatformFoundationWatchOS.h>
```

For Swift:

```
import IBMMobileFirstPlatformFoundationWatchOS
```

Setting up MobileFirst security for the iPhone app and the watchOS 2 app:

You can set up MobileFirst security for your iPhone app and watchOS 2 app by registering each as a separate target on the IBM MobileFirst Platform Server.

Before you begin

For this example, you must have already created:
- An Xcode project with the MobileFirst frameworks installed using both a main app and a watchkit extension, each registered separately on the MobileFirst Server. See "Setting up watchOS 2 development in Xcode" on page 7-48.
- An adapter with a defined scope, and two security checks: one for username/password and one for a pin code. For more information on the configuring the security see "Security checks" on page 7-281.

.

About this task

The Apple Watch and iPhone devices differs physically. Therefore the security checks for each must be appropriate for the available input devices. For example, the Apple Watch is limited to a number pad and does not allow the usual username/password security check. Therefore access to protected resources on the server could be enabled using a pin code. Because of these and similar differences, it is necessary to apply different security checks for each target.

Below is one example of creating an app with both an iPhone and an Apple Watch target. This architecture allows each to have its own security check. The differing security checks are just examples of how you can design features for each target. Additional security checks might be available.

Procedure
1. Determine the scope and security checks defined by the protected resource. See "Security-checks configuration" on page 7-297.
2. In the IBM MobileFirst Platform Operations Console:
   a. Ensure that both apps are registered on the server:
      - `com.worklight.[project_name]`
      - `com.worklight.[project_name].watchkitextension`
   b. Map the `scopeName` to the defined security checks:
      - For `com.worklight.[project_name]` map it to the username/password check.
      - For `com.worklight.[project_name].watchkitapp.watchkitextension` map it to the pin code security check.

Results

The Xcode project now contains a main app and a watchOS 2 app, each with its own security check. For more results see the watchOS Tutorial.

WatchOS 2 limitations:

The MobileFirst SDK for watchOS 2 does not support all MobileFirst features.

Limitations

The optional frameworks that add features to the MobileFirst app are not provided for watchOS 2 development. Some other features are limited by constraints imposed by the watchOS 2 or Apple Watch device.

Table 7-10. watchOS 2 non-supported features

feature	
openSSL	not supported
JSONStore	no supported
push	not supported
message alerts displayed by the MobileFirst code	not supported
application-authenticity validation	not compatible with bitcode, and therefore not supported
remote disable/notify	requires customization (see below)
usernames/password security check	use the pin code security check (see "Setting up MobileFirst security for the iPhone app and the watchOS 2 app" on page 7-51)

Remote disable/notify

With the IBM MobileFirst Platform Operations Console, you can configure the IBM MobileFirst Platform Server to disable access (and return a message) to client applications based on the version they are running (see "Remotely disabling application access to protected resources" on page 10-17). Two options provide default UI alerts:

- when the app is active but a messages is sent: **Active and Notifying**
- when the app is outdated and access is denied: **Access Denied**

For watchOS 2:

- To see messages where the app is set to **Active and Notifying**, a custom remote disable challenge handler must be implemented and registered. The custom challenge handler must be initialized with the security check `wl_remoteDisableRealm`.
- In the case where the access is disabled (**Access Denied**) the client app receives an error message in the failure callback or request delegate handler. The developer can decide how to handle the error, either notifying the user through the UI or writing to the log. In addition the above method of creating a custom challenge handler can be used.

Developing native applications in Android Studio

To create an IBM MobileFirst Platform Foundation Android app add the necessary SDK files to your Android Studio project and register your app.

To develop a native application, add the MobileFirst SDK files to your Android Studio project. Then register your app using IBM MobileFirst Platform Operations Console or the MobileFirst Platform CLI. Develop and compile your app.

You can set up your Android Studio to include the MobileFirst by using Gradle or by using a manual or remote method of retrieving the files. For more information, see "Methods of setting up your environment" on page 7-53.

Once you have the IBM MobileFirst Platform Foundation SDK files set up in your Xcode project, you can register your app on the server. For details, see "Registering Android applications to MobileFirst Server" on page 7-59.

Note: The MobileFirst APIs cannot be activated from within an Android Service.

The following topics show you how to set up an initial project and start developing.

Methods of setting up your environment

You can prepare your environment for developing MobileFirst applications by copying several files to your Android Studio project.

You can add the IBM MobileFirst Platform Foundation for Android SDK files to your Android Studio project from either remote or local files using Gradle.

Once you have added your SDK to your Android project and imported the necessary classes into your code, you can add MobileFirst functionality to your app.

Setting up Android Studio projects with Gradle:

Before you begin

Ensure that you set up Android Studio and the Android SDK properly. For more information about how to set up your system, see Android Studio Overview.

Note: MobileFirst SDK is compatible with Android version Ice Cream Sandwich (API level 14) and later.

About this task

The documented and supported development environment for Android applications with MobileFirst SDK is now Android Studio. To develop a new Android application with Android Studio and MobileFirst SDK, follow these steps.

Procedure

1. If you do not already have one, create an Android application in Android Studio by using **File > New > New Project** wizard. Make sure the project compiles without error.

   Note: There are two versions of the `build.gradle` file created, one in the main project folder and one in the `\apps` folder.
2. Make sure the `[project]\build.gradle` file has the `jcenter()` in list of repositories in the `allprojects{}` closure.
   ```
   allprojects {
     repositories {
       jcenter()
       // other repositories
     }
   }
   ```
   The following example shows a sample `[project]\build.gradle` file created by the Android Studio wizard:
   ```
   // Top-level build file where you can add configuration options common to all sub-projects/modules.

   buildscript {
   ```

```
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.3.0'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

The actual contents of your file can vary, depending on whether other repositories or dependencies have been added.

3. Add the following packaging options within your `android{}` closure in the `app\build.gradle` file:

```
packagingOptions {
  pickFirst 'META-INF/ASL2.0'
  pickFirst 'META-INF/LICENSE'
  pickFirst 'META-INF/NOTICE'
}
```

4. Depending on whether you are installing `aar` files from a local copy or accessing them remotely, add the following lines to your `app\build.gradle` file.
   a. If you are installing from remote copies of the files add this line to the dependencies closure.

```
compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundation:8.0.+'
```

   Note: In this example the latest version of 8.0 is imported. If you want to import a specific version such as 8.0.2016021411, replace with the MobileFirst version number you are using, including the major, minor, and patch numbers. The patch number is in the format *YYYYMMDDHH*. For example:

```
compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundation:8.0.2016021411'
```

   The following is an example of an `app/build.gradle` file for adding the SDK from remote files:

```
applyplugin: 'com.android.application'

repositories {
    jcenter()
}
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.example.myname.myapplicationandroidgradle"
        minSdkVersion 23
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    packagingOptions {
        pickFirst 'META-INF/ASL2.0'
        pickFirst 'META-INF/LICENSE'
        pickFirst 'META-INF/NOTICE'
    }
}
dependencies {
compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundation:8.0.+'
    }
}
```

b. Remove the `proguardFiles` line from the `buildTypes` enclosure and save the file. Proguard is not supported in IBM MobileFirst Platform Foundation V8.0.0. See Obfuscating Android code with ProGuard.

c. If you are installing from local files:

Add the following to the dependencies closure.

```
compile(name:'ibmmobilefirstplatformfoundation', ext:'aar')
```

Add a `repositories` enclosure:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

Note: To acquire the necessary SDK files see "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26. Copy the relevant `aar` files to the `app\libs` folder.
The following is an example of an `app\build.gradle` file for adding the SDK from local files:

```
apply plugin: 'com.android.application'

repositories {
    flatDir {
        dirs 'libs'
    }
}
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.example.myname.myapplicationandroidgradle"
        minSdkVersion 23
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

    packagingOptions {
        pickFirst 'META-INF/ASL2.0'
        pickFirst 'META-INF/LICENSE'
        pickFirst 'META-INF/NOTICE'
    }
}
dependencies {
    compile(name:'ibmmobilefirstplatformfoundation', ext:'aar')
}
```

5. The imported SDK does not include the Javadocs. To add the Javadocs to your project see "Registering Javadocs to an Android Studio Gradle project" on page 7-57.

6. For information on adding additional features see "Adding the optional MobileFirst components with Gradle" on page 7-56.

7. Add the following lines to the `AndroidManifest.xml` file of your native app for Android:

   - `<activity android:name="com.worklight.wlclient.ui.UIActivity" />`

     This line adds the ability for a designated MobileFirst UI activity to run in the user application.

Note: If you want to automate this process you can add the following task to the `app\build.gradle` file:

```
task(addUIActivity) << {
def manifestFile = file("src/main/AndroidManifest.xml")
def manifestText = manifestFile.getText()
if(!manifestText.contains("com.worklight.wlclient.ui.UIActivity")) {
     def pattern =  Pattern.compile("\\</application\\>")
     def matcher =  pattern.matcher(manifestText)
     def manifestContent = matcher.replaceFirst("<activity android:name=\"com.worklight.wlclient.ui.UIActivity\"/>\n</application>")
      manifestFile.write(manifestContent)
  }
}
preBuild.dependsOn addUIActivity
```

- `<uses-permission android:name="android.permission.INTERNET" />`

  This line adds internet access permissions to the user application.

8. Rebuild your application.

Results

You can now start developing your native Android application with the IBM MobileFirst Platform Foundation SDK.

What to do next

Before you can access server resources, you must register your app. See "Registering Android applications from the MobileFirst Platform CLI" on page 7-59. For details about the `mfpclient.plist` file see "Android client properties file" on page 7-62. Once the app is registered you can write some initial code for testing the server connection ("Some initial code for accessing the server" on page 7-63.

Adding the optional MobileFirst components with Gradle:

You can add more MobileFirst features to your Android application with Gradle.

Before you begin

Prerequisite: An existing Android Studio project with the app Gradle file set up to add the core MobileFirst SDK. See "Setting up Android Studio projects with Gradle" on page 7-53.

About this task

To add additional features, add the optional `aar` files to your Android Studio project. Some additional steps for setting up Android Studio to support your feature may exist.

These are the optional `aar` files:

Table 7-11. Android SDK

aar	**feature**
`ibmmobilefirstplatformfoundationpush.aar`	Push notifications. For more information on setup see "Setting up push notifications" on page 7-253.
`ibmmobilefirstplatformfoundationjsonstore.jar`	JSONStore. For more information on setup see "JSONStore" on page 7-134.

Procedure

1. For each aar to be added, add entries to the build.gradle file.
   a. For accessing and installing aar remotely, add a compile entry to the dependencies enclosure in the app\build.gradle file. There should be an existing compile entry for the main MobileFirst aar file.

      ```
      dependencies {
          compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundation:8.0.+'
          compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundationpush:8.0.+'
          compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundationjsonstore:8.0.+'
      }
      ```

      Note: In this example the latest version of 8.0 is imported. If you want to import a specific version such as 8.0.2016021411, replace with the MobileFirst version number you are using, including the major, minor, and patch numbers. The patch number is in the format *YYYYMMDDHH*. For example:

      ```
      compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundation:8.0.2016021411'
      ```

   b. For installing locally available aar files, copy the optional aar file to the libs folder. Add a line to the dependency enclosure for each aar file needed. This example adds all the optional and required SDKs.

      ```
      dependencies {
          compile(name:'ibmmobilefirstplatformfoundation', ext:'aar')
          compile(name:'ibmmobilefirstplatformfoundationjsonstore', ext:'aar')
          compile(name:'ibmmobilefirstplatformfoundationpush', ext:'aar')
      }
      ```

      Note: For information on obtaining the available optional aar files see "Acquiring the MobileFirst SDK from the MobileFirst Operations Console" on page 7-26.
2. Rebuild your application.
3. The imported SDK includes the Javadocs but they need to be linked in your Android Studio project. For details see "Registering Javadocs to an Android Studio Gradle project."

Results

You can now start developing your native IBM MobileFirst Platform Foundation Android app with additional features.

Registering Javadocs to an Android Studio Gradle project:

Before you begin

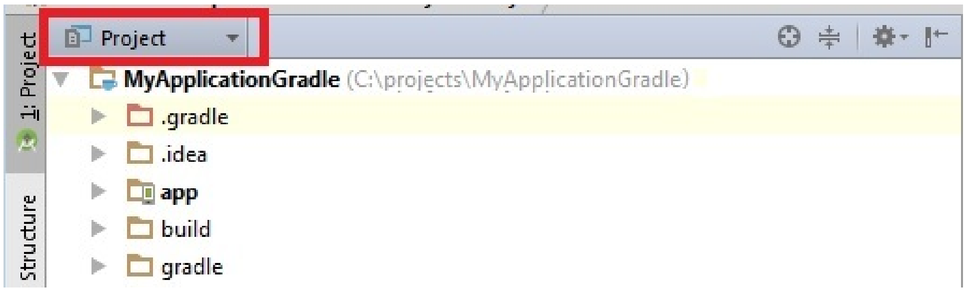
This task is necessary after you have installed the MobileFirst SDK using Gradle.

About this task

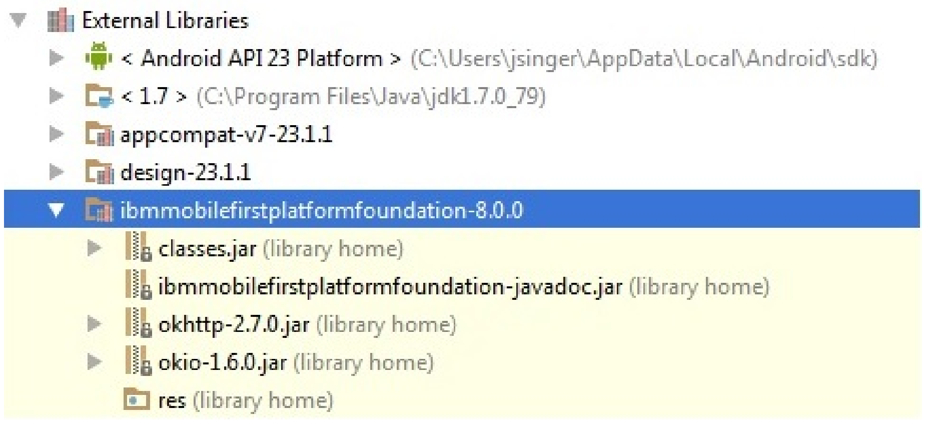
The MobileFirst Android Javadocs are included in the *.aar files imported by Gradle. However you need to link them to their relevant library in Android Studio.

Procedure

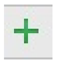
1. In Android Studio make sure you are in the **Project** view, accessible from the pull-down menu above the navigation bar:



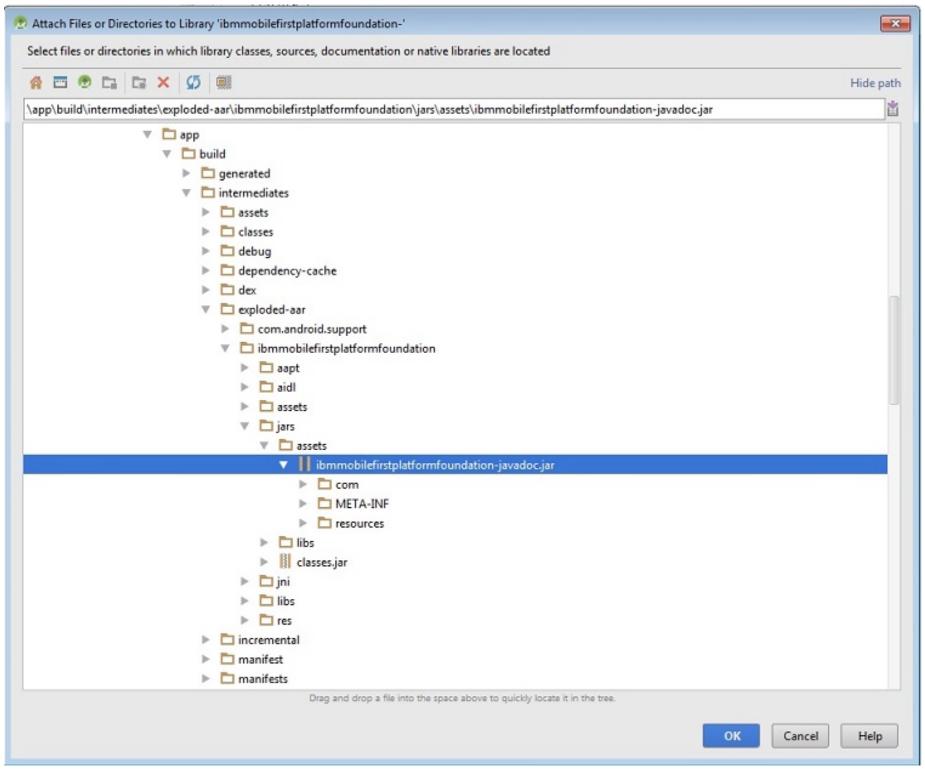
a. Find the library name under the **External Libraries** node (the Javadoc file appears below it).


External Libraries
< Android API 23 Platform > (C:\Users\jsinger\AppData\Local\Android\sdk)
< 1.7 > (C:\Program Files\Java\jdk1.7.0_79)
appcompat-v7-23.1.1
design-23.1.1
ibmmobilefirstplatformfoundation-8.0.0
classes.jar (library home)
ibmmobilefirstplatformfoundation-javadoc.jar (library home)
okhttp-2.7.0.jar (library home)
okio-1.6.0.jar (library home)
res (library home)

b. Right-click on the library name . Choose **Library Properties**.

2. From the **Library Properties** dialog select the  .

   a. Navigate to the downloaded Javadoc JAR file (`ibmmobilefirstplatformfoundation-javadoc.jar`) under `..\app\build\intermediates\exploded-aar\ibmmobilefirstplatformfoundation\jars\assets\` and select it.


Attach Files or Directories to Library 'ibmmobilefirstplatformfoundation-'

Select files or directories in which library classes, sources, documentation or native libraries are located

\app\build\intermediates\exploded-aar\ibmmobilefirstplatformfoundation\jars\assets\ibmmobilefirstplatformfoundation-javadoc.jar

```
app
  build
    generated
    intermediates
      assets
      classes
      debug
      dependency-cache
      dex
      exploded-aar
        com.android.support
        ibmmobilefirstplatformfoundation
          aapt
          aidl
          assets
          jars
            assets
              ibmmobilefirstplatformfoundation-javadoc.jar
                com
                META-INF
                resources
            libs
            classes.jar
          jni
          libs
          res
      incremental
      manifest
      manifests
```

Drag and drop a file into the space above to quickly locate it in the tree.

b. Click **OK**.

Results

The Javadocs will now be available within your project.

Registering Android applications to MobileFirst Server

Register your Android application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app. You can register your app using the IBM MobileFirst Platform Command Line Interface (CLI) or the IBM MobileFirst Platform Operations Console.

Registering Android applications from the MobileFirst Platform CLI:

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to register your Android application to an instance of MobileFirst Server.

Before you begin

- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see “Installing the MobileFirst Platform CLI” on page 7-15.
- You must have an instance of MobileFirst Server running. The server can be running locally, or it can be a remote server, but it must be reachable from your local computer. For more information, see “Setting up the MobileFirst Development Server” on page 7-12, “Installing the IBM MobileFirst Platform Foundation Developer Kit” on page 7-10, or “Installing IBM MobileFirst Platform Server” on page 6-2.
- You must have an Android application on your local computer.

About this task

Once you have the client side of your Android application initially defined, you can prepare for further development tasks by registering it to a MobileFirst Server.

Procedure

1. Check that the target MobileFirst Server is up and running.
2. Navigate to the directory that contains your app, or one of its subdirectories.
3. Register your app to the server. Use one of the following procedures:
   - To register the app to the default server, run the following command:
     ```
     mfpdev app register
     ```
     Note: If you have not previously set a default server and a MobileFirst Server is running on the local system, this command registers the app to the local MobileFirst Server, and this server is made the default.
   - To register your app to a server that is not the default server:
     a. Create a server profile by running the **`mfpdev server add`** command. For example:
        ```
        mfpdev server add Server1 -url https://company.mobile.com:9080 -login admin -password secre
        ```
        For more information about the **`mfpdev server add command`**, run `mfpdev help server add`.
     b. To register your app to the server that you just defined, run the **`mfpdev app register`** command, and specify the server profile that you just created. For example:
        ```
        mfpdev app register Server1
        ```

   For more information about this command, including optional parameters, run **`mfpdev help app register`**.

Results

The app is registered to the target server. Data about the app that is obtained from the platform properties file (`AndroidManifest.xml`) such as application name, version number, and app ID is sent to the server. If the client properties file `/app/src/main/assets/mfpclient.properties` already exists, it is updated with the value of the server's URL. If the file did not exist, a `/app/src/main/assets/mfpclient.properties` file that includes the server's URL is created at the root of your project.

What to do next

You can proceed with other development tasks that depend on the MobileFirst Server. For example, you can preview your app, test your app's security features, and manage your app from the MobileFirst Operations Console.

Registering Android applications from the MobileFirst Operations Console:

Register your Android application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app.

Before you begin

You must have the IBM MobileFirst Platform Operations Console running on the MobileFirst Server targeted for registration. For more information, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.

About this task

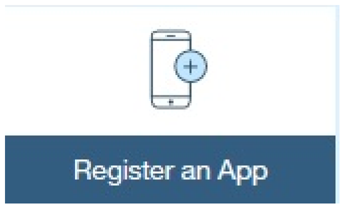
You can register an app on the server before or after setting up the Android Studio environment (see "Methods of setting up your environment" on page 7-53). You must register your app to the server before developing code that accesses server resources.

Procedure

1. In the MobileFirst Operations Console, navigate to the Register Application page by using one of the following methods:
   - In the navigation sidebar, select **New** next to **Appilcations**.

     Applications (0)
     New

   - From the **Dashboard**, select **Register an App**.

2. On the Register an Application page, select the **Android** option.

3. Fill in the values for the application that you are registering.

   If you created your app by using the Android Studio wizard, the version number and package name and are defined in the `app\build.gradle` file in your Android Studio project, in the `defaultConfig` enclosure:

   ```
   defaultConfig {
       applicationId "com.example.myapplication"
       minSdkVersion 23
       targetSdkVersion 23
       versionCode 1
       versionName "1.0"
   }
   ```

   - **Application Name**: This is for display and can be any convenient value. It is optional.
   - **Package**: The **`applicationId`** value specified in your `app/build.gradle`. For more information, see http://tools.android.com/tech-docs/new-build-system/applicationid-vs-packagename.
   - **Version**: The **`versionName`** value found in your `app/build.gradle`.

     If these parameters do not exist in the `app\build.gradle`, you can find them in the `AndroidManifest.xml` file.

```
<manifest android:hardwareAccelerated="true" android:versionCode="2" android:versionName="0.0.2" ...>
```

4. Click the **Register application** button. From the main **Dashboard** page, your application is now listed under the default **mfp** runtime.

Note: The `mfp` runtime is the default value for the **`WLServerContext`** parameter in the `mfpclient.properties` file below.

5. Click the application name to display the main configuration page for your app. The main page for your app displays different configuration options for the server-side registration of your app.
6. Click the **Configuration Files** tab. The Client Configuration File section displays a template for creating your `mfpclient.properties` file. This file is used for connecting the client app to the server. For information on populating the values in the file and copying it to your Android Studio project, see "Android client properties file."

Results

The app is registered on the target server.

What to do next

To complete the client-server registration, you must complete the required properties in the `mfpclient.properties` file and copy it to your Android Studio project. For more information, see "Android client properties file."

Android client properties file

The `mfpclient.properties` file defines the properties that your native app for Android requires to use the MobileFirst native API for Android.

The `mfpclient.properties` file is created when you register your Android app on the MobileFirst Development Server (see "Registering Android applications to MobileFirst Server" on page 7-59). If you register your using the IBM MobileFirst Platform Operations Console you must create the file manually and place it within your Android Studio project.

The following table lists the properties of the `mfpclient.properties` file, their descriptions, and possible values.

Table 7-12. Properties and values of the `mfpclient.properties` file.

Property	Description	Example values
`wlServerProtocol`	The communication protocol with the MobileFirst Server.	`http` or `https`
`wlServerHost`	The host name of the MobileFirst Server.	`192.168.1.63` If the MobileFirst Server host has an IPV6
`wlServerPort`	The port of the MobileFirst Server.	`9080`
`wlServerContext`	The server context. This value can be seen in the MobileFirst Operations Console dashboard. It is the value of the app's runtime. By default the values is `/mfp/`.	`/mfp/`
`wlPlatformVersion`	The MobileFirst version	`8.0.20160214`
`languagePreferences`	Preferred language.	`en`

Below is an example `mfpclient.properties` file.

```
wlServerProtocol=http
wlServerHost=9.148.49.221
wlServerPort=9080
wlServerContext=/mfp/
wlPlatformVersion=8.0.20160214
languagePreferences=en
```

Note: You can create this file manually and place it in the `[project]\app\src\main\assets` folder of your Android Studio project folder. Add the `assets` folder if it does not exist under the `main` folder. When you register the app by using MobileFirst Platform CLI the `mfpclient.properties` file is created, along with the correct values, and placed in the `assets` folder.

Some initial code for accessing the server

A simple startup process for Android-based applications for accessing the server is described here.

The main activity

Before you can access the server resources, you must register the app on the server. For more information, see "Registering Android applications to MobileFirst Server" on page 7-59.

You must also deploy an adapter. For more information about using adapters, see "Client access to adapters" on page 7-231.

This example is based on the sample code created in "Setting up Android Studio projects with Gradle" on page 7-53.

Import the MFP client API and create an instance

Some imports are required to run this example:

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import com.worklight.common.Logger;
import com.worklight.wlclient.api.WLAccessTokenListener;
import com.worklight.wlclient.api.WLAuthorizationManager;
import com.worklight.wlclient.api.WLClient;
import com.worklight.wlclient.api.WLFailResponse;
import com.worklight.wlclient.api.WLResourceRequest;
import com.worklight.wlclient.api.WLResponse;
import com.worklight.wlclient.api.WLResponseListener;
import com.worklight.wlclient.auth.AccessToken;
import com.worklight.wlclient.auth.WLAuthorizationManagerInternal;

import java.net.URI;
import java.net.URISyntaxException;
```

The `WLClient` is used throughout the application to connect to the server.

The resource request calls the `getBankBalance` adapter and writes messages to the log for success or failure.

Replace the entire **MainActivity** with the code that follows.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        WLClient client = WLClient.createInstance(this);
        URI adapterPath = null;

        try {
           adapterPath = new URI("/adapters/javaAdapter/users/getBankBalance");
           } catch (URISyntaxException e) {
            e.printStackTrace();
            }

        WLResourceRequest request = new WLResourceRequest(adapterPath, WLResourceRequest.GET);
        request.send(new WLResponseListener() {
        @Override
            public void onSuccess(WLResponse wlResponse) {
                Log.i("MobileFirst Quick Start", "Success: " + wlResponse.getResponseText());
            }
        @Override
            public void onFailure(WLFailResponse wlFailResponse) {
                Log.i("MobileFirst Quick Start", "Failure: " + wlFailResponse.getErrorMsg());
            }
        });
    }
```

Accessing the server without deploying an adapter

To test the server connectivity without deploying an adapter, request an access token. If no security checks were added to your app, and the server can be accessed, the token can be obtained. The token request is sent without a scope:

```
WLAuthorizationManager.getInstance().obtainAccessToken("", new MyObtainAuthorizationHeaderListener());
```

The entire `MainActivity` consists of creating the `WLClient` and requesting the access token:

```
public class MainActivity extends AppCompatActivity {

@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        WLClient client = WLClient.createInstance(this);
        URI adapterPath = null;
        Logger.setContext(this);
        WLAuthorizationManager.getInstance().obtainAccessToken("", new MyObtainAuthorizationHeaderListener());
    }
```

The `MyObtainAuthorizationHeaderListener` listener returns success or error messages.

```
class MyObtainAuthorizationHeaderListener implements WLAccessTokenListener {

    @Override
    public void onSuccess(AccessToken accessToken) {
        Log.i("MobileFirst Quick Start", " Success ");
        return;
    }

    @Override
    public void onFailure(WLFailResponse wlFailResponse) {
        String errorMsg=wlFailResponse.getErrorMsg();
        if (errorMsg != null)
            errorMsg=errorMsg.replace("\n","").replace("\r","").replace("\t","");
        Log.i("MobileFirst Quick Start", "failure: " + errorMsg);
        return;
    }
}
```

The error or success message is printed to the Android monitor pane in Android Studio.

After your app connects successfully to the server, you can continue to develop your app.

Developing native C# applications for Windows 10 Universal Windows Platform and Windows 8 Universal

To develop a native C# application for Windows 10 Universal Windows Platform (UWP) and Windows 8 Universal, you must create a Visual Studio project and then add the IBM MobileFirst Platform Foundation SDK files to it.

When you create native API applications, you can choose to create a Windows 8 Universal environment that supports either desktops and tablets or phones or a Windows 10 UWP environment. The contents of the associated application descriptor file reflect your choice.

Methods of setting up your environment

You can prepare your environment for developing MobileFirst applications by installing the `IBM.MobileFirstPlatformFoundation` NuGet package into your Visual Studio project.

Adding the MobileFirst SDK manually:

You can prepare your environment for developing MobileFirst applications by getting the framework and library files manually. The IBM MobileFirst Platform Foundation SDK for Windows 8 and Windows 10 Universal Windows Platform (UWP) is also available from NuGet.

About this task

You can add MobileFirst functionality to an existing application, or upgrade from an earlier version of the MobileFirst SDK, by manually adding the SDK files to your application. Follow the outlined procedure to obtain the required SDK files and to prepare your development environment to build MobileFirst applications.

Procedure

1. Get the MobileFirst SDK from MobileFirst Operations Console.You can refer to Acquiring the MobileFirst SDK from the MobileFirst Operations Console for detailed instructions.
2. Extract the contents of the downloaded SDK obtained in step 1.
3. Open the Windows Universal native project in Visual Studio. Perform the following steps.
   a. Select **Tools > NuGet Package Manager > Package Manager Settings**.
   b. Select **Package Sources** option. Click **+** icon to add new package source.
   c. Provide a name for the package source (for example: `windows8nuget`).
   d. Navigate to the MobileFirst SDK folder that was downloaded and extracted. Click **OK**.
   e. Click **Update** and then click **OK**.
   f. Right-click the **Solution project-name** in **Solution explorer** tab, which is to the right corner of the screen.
   g. Select **Manage NuGet Packages for Solutions > Online > windows8nuget**.
   h. Click **Install** option. You get the option to **Select Projects**.
   i. Ensure that all the check boxes are checked. Click **OK**.

   NuGet package gets installed into your project.

Results

Your Windows native project is now ready for development of MobileFirst applications.

In your application, you must import the headers for the IBMMobileFirstPlatformFoundation framework by using IBM.Worklight APIs.

Adding the MobileFirst SDK by using NuGet:

You can prepare your environment for developing MobileFirst applications by getting the framework and library files through installing `IBM.MobileFirstPlatformFoundation` package from NuGet. The IBM MobileFirst Platform Foundation SDK for Windows 8 and Windows 10 Universal Windows Platform (UWP) is available from NuGet.

About this task

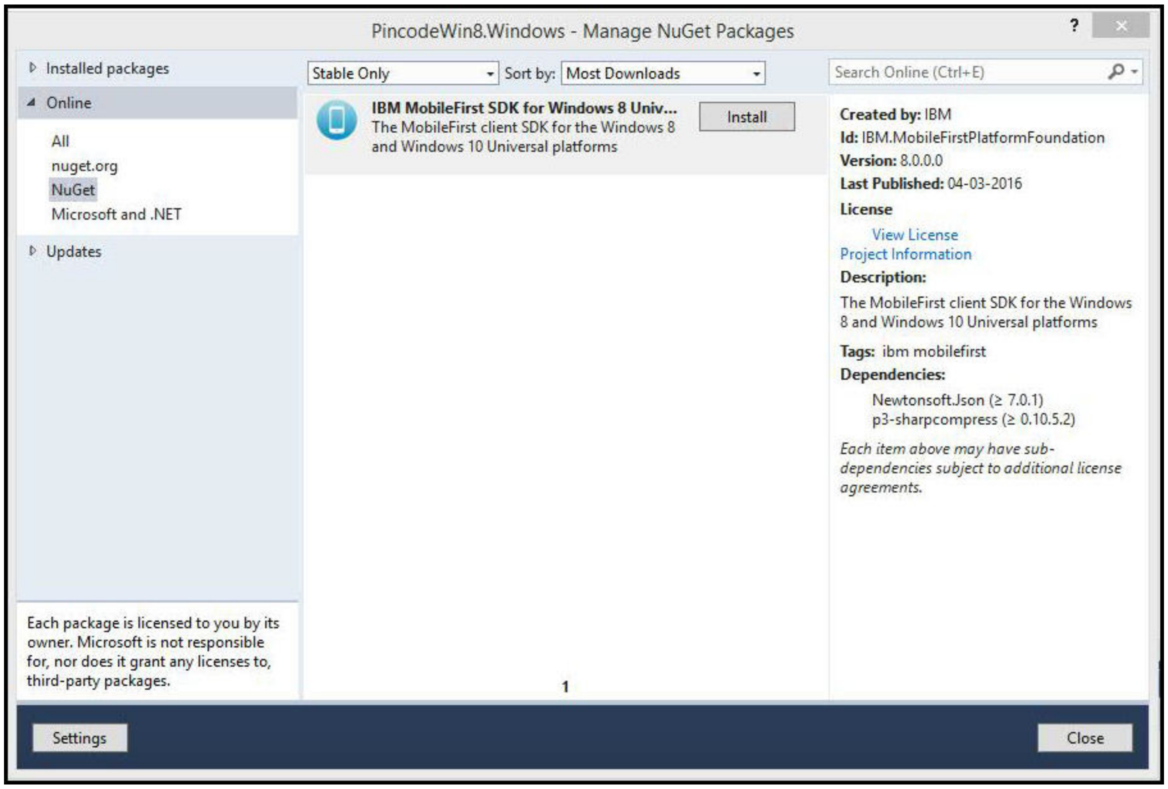
Information on packages on NuGet is available on the Nuget Packages page at `https://www.nuget.org/packages`. To add the MobileFirst SDK to your Visual Studio Project, follow these steps:

Procedure

1. Create a Visual Studio C# Project for Windows Universal or open an existing project or solution.
2. Select **Tools** > **NuGet Package Manager** > **Package Manager Console**.
3. Choose the project where you want to install the MobileFirst SDK.
4. Install the MobileFirst SDK by running the `Install-Package IBM.MobileFirstPlatformFoundation` command.

   Or,

   You can also add the MobileFirst SDK to your Visual Studio Project by right-clicking the **References** tab of your project and selecting **Manage NuGet Packages**. Search for `IBM.MobileFirstPlatformFoundation` and click **Install**.


PincodeWin8.Windows - Manage NuGet Packages

IBM MobileFirst SDK for Windows 8 Univ...
The MobileFirst client SDK for the Windows 8 and Windows 10 Universal platforms

Created by: IBM
Id: IBM.MobileFirstPlatformFoundation
Version: 8.0.0.0
Last Published: 04-03-2016
License
View License
Project Information
Description:
The MobileFirst client SDK for the Windows 8 and Windows 10 Universal platforms
Tags: ibm mobilefirst
Dependencies:
Newtonsoft.Json (≥ 7.0.1)
p3-sharpcompress (≥ 0.10.5.2)
Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Figure 7-4. Installing MobileFirst SDK using the Manage NuGet Packages

5. Optional: You can also add optional MobileFirst components by getting the framework and library files through installing package from NuGet. For more information, see Adding the optional MobileFirst components by using NuGet.

Results

You can now start developing your native Windows Universal applications with the MobileFirst SDK.

Note: There is a known limitation with MobileFirst Windows 10 UWP apps when the MobileFirst SDK is installed through the NuGet package.

Adding the optional MobileFirst components by using NuGet:

You can prepare your environment for developing MobileFirst applications with optional components such as MobileFirst push by getting the framework and library files through installing `IBM.MobileFirstPlatformFoundationPush` package from NuGet. The MobileFirst SDK for Windows 8 and Windows 10 Universal Windows Platform (UWP) is a prerequisite and is also available from NuGet.

About this task

Information on packages on NuGet is available on the Nuget Packages page at `https://www.nuget.org/packages`. To add the optional MobileFirst push to your Visual Studio Project, follow these steps:

Procedure

1. Create a Visual Studio C# Project for Windows Universal or open an existing project or solution.
2. Select **Tools** > **NuGet Package Manager** > **Package Manager Console**.
3. Choose the project where you want to install the MobileFirst push component.
4. Install the MobileFirst push component by running the `Install-Package IBM.MobileFirstPlatformFoundationPush` command.

   Or,

   You can also add the MobileFirst push component to your Visual Studio Project by right-clicking the **References** tab of your project and selecting **Manage NuGet Packages**. Search for `IBM.MobileFirstPlatformFoundationPush` and click **Install**.

Results

You can now develop your native Windows Universal applications with the capability to send push notifications by using the MobileFirst push component.

Related links

"Adding the MobileFirst SDK by using NuGet" on page 7-66
You can prepare your environment for developing MobileFirst applications by getting the framework and library files through installing `IBM.MobileFirstPlatformFoundation` package from NuGet. The IBM MobileFirst Platform Foundation SDK for Windows 8 and Windows 10 Universal Windows Platform (UWP) is available from NuGet.

Registering Windows applications to MobileFirst Server

Register your Windows application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app.

About this task

You can register your app by using the IBM MobileFirst Platform Command Line Interface (CLI) or the IBM MobileFirst Platform Operations Console.

Registering Windows applications from the MobileFirst Platform CLI:

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to register your Windows application to an instance MobileFirst Server.

Before you begin

- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have an instance of MobileFirst Server running. The server can be running locally, or it can be a remote server, but it must be reachable from your local computer. For more information, see "Setting up the MobileFirst Development Server" on page 7-12, "Installing the IBM MobileFirst Platform Foundation Developer Kit" on page 7-10, or "Installing IBM MobileFirst Platform Server" on page 6-2.
- You must have a native Windows application on your local computer.

About this task

Once you have the client side of your Windows application initially defined, you can prepare for further development tasks by registering it to a MobileFirst Server.

Registration occurs for only the app that corresponds to the current working directory in which you run the **mfpdev app register** command. If the current working directory is a Windows 8.1 Universal project that contains both Windows 8.1 desktop and Windows Phone 8.1 then both the desktop and Windows Phone projects are registered. If the current working directory is the Windows or WindowsPhone directory under the Windows 8.1 Universal project, or one of their subdirectories, then registration occurs for only the project that corresponds to the directory that you are in.

Procedure

1. Check that the target MobileFirst Server is up and running.
2. Navigate to the directory that contains your app, or one of its subdirectories.
3. Register your app to the server. Use one of the following procedures:
   - To register the app to the default server, run the following command:
     ```
     mfpdev app register
     ```
     Note: If you have not previously set a default server and a MobileFirst Server is running on the local system, this command registers the app to the local MobileFirst Server, and this server is made the default.
   - To register your app to a server that is not the default server:
     a. Create a server profile by running the **mfpdev server add** command. For example:
        ```
        mfpdev server add Server1 -url https://company.mobile.com:9080 -login admin -password se
        ```
        For more information about the **mfpdev server add command**, run `mfpdev help server add`.
     b. To register your app to the server that you just defined, run the **mfpdev app register** command, and specify the server profile that you just created. For example:
        ```
        mfpdev app register Server1
        ```

   For more information about this command, including optional parameters, run **mfpdev help app register**.

Results

The app is registered to the target server. Data about the app that is obtained from its platform properties file (`.appxmanifest`) such as application name, version number, and app ID is sent to the server. If the root client properties file (`mfpclient.resw`) for each registered Windows platform exists, it is updated with the value of the server's URL. If the file did not exist, a `mfpclient.resw` file is created that includes the server's URL. The client properties files are located as follows, depending on your specific Windows platform:

- Windows 10 UWP and stand-alone Windows 8.1 projects (either Windows 8.1 desktop or Windows Phone 8.1):`/strings/mfpclient.resw`

For Windows 8.1 Universal projects, two client properties files are created or updated:

- Windows Phone 8.1: `/WindowsPhone/strings/mfpclient.resw`

- Windows 8.1 desktop:`/Windows/strings/mfpclient.resw`

What to do next

If the `mfpclient.resw` file was created when you ran **`mfpdev app register`**, then the file is not linked into your Windows project in your IDE and you need to link it. If the `mfpclient.resw` was created by using NuGet, then the file is already linked into your project, but you need to update the properties in the file to correspond to your target MobileFirst Server. For more information, see "Client property file for Windows 10 Universal Windows Platform and Windows 8 Universal" on page 7-72.

You can proceed with other development tasks that depend on the MobileFirst Server. For example, you can preview your app, test your app's security features, and manage your app from the MobileFirst Operations Console.

Registering Windows applications from the MobileFirst Operations Console:

You can use the IBM MobileFirst Platform Operations Console to register your Windows application to an instance of MobileFirst Server.

About this task

You must have the IBM MobileFirst Platform Operations Console running on the MobileFirst Server targeted for registration. For more information, see "The IBM MobileFirst Platform Foundation Developer Kit" on page 7-9.

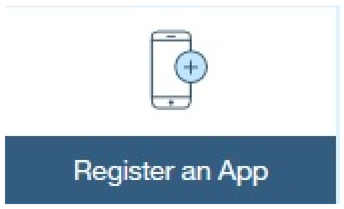
You can register an app on the server before or after setting up the Visual Studio environment (see "Methods of setting up your environment" on page 7-65). You must register your app to the server before developing code that accesses server resources.

Procedure

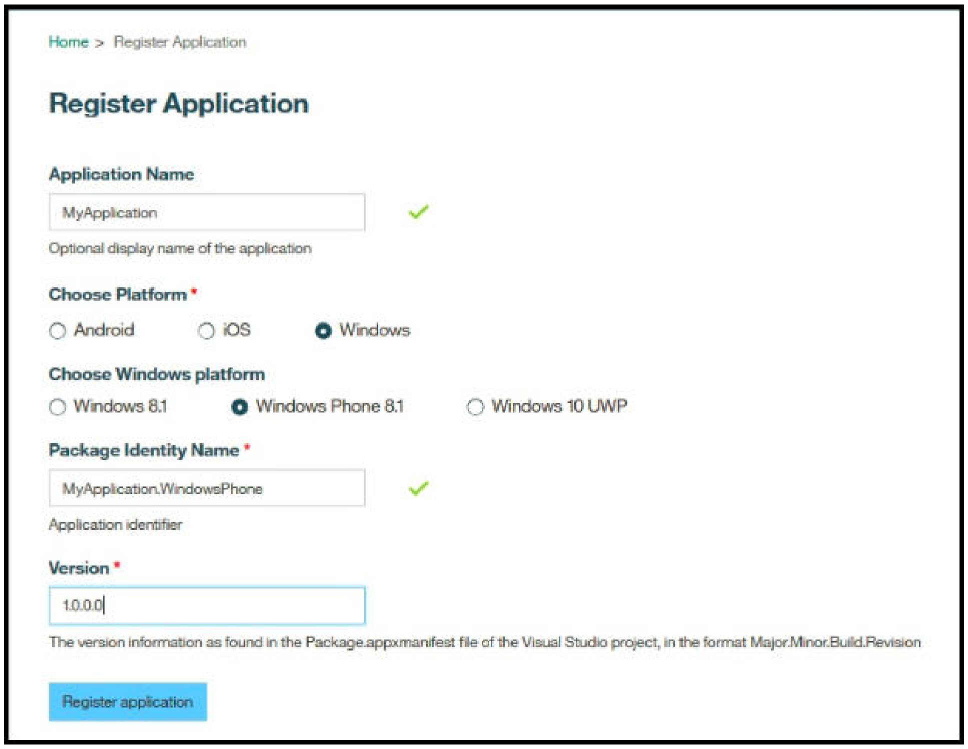
1. In the MobileFirst Operations Console, navigate to the Register Application page by using one of the following methods:
   - In the navigation sidebar, select **New** next to **Appilcations**.

     Applications (0)
New

   - From the **Dashboard**, select **Register an App**.

2. On the **Register Application** page, select the **Windows** option and then select the appropriate Windows platform you are working on.


Home > Register Application

Register Application

Application Name

MyApplication ✓

Optional display name of the application

Choose Platform *

○ Android ○ iOS [✓ Windows]

Choose Windows platform

○ Windows 8.1 [✓ Windows Phone 8.1] ○ Windows 10 UWP

Package Identity Name *

MyApplication.WindowsPhone ✓

Application identifier

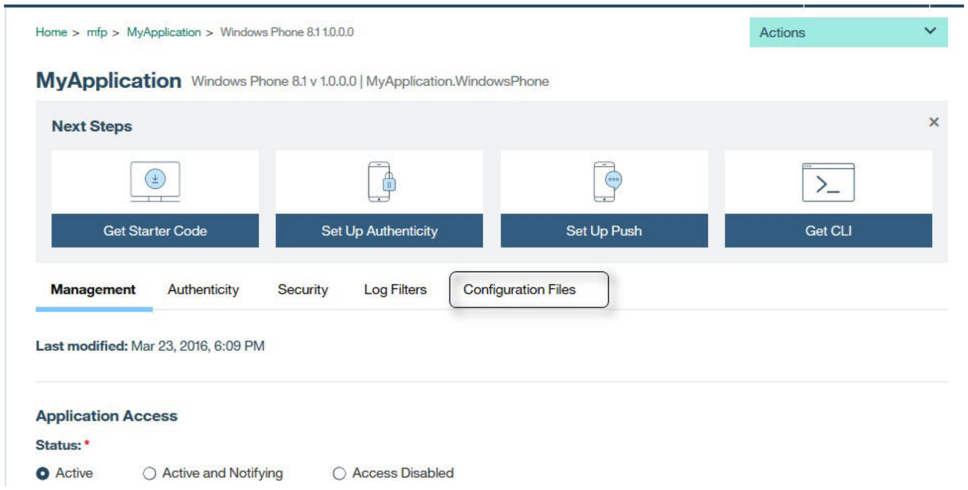
Version *

1.0.0.0

The version information as found in the Package.appxmanifest file of the Visual Studio project, in the format Major.Minor.Build.Revision

3. Update the following values for the application you are registering:
   - **Application Name**: This is for display and can be any convenient value. It is optional.
   - **Package identity name**: This is typically the name of your Visual Studio project and can be found in the `Package.appxmanifest` file in your Visual Studio project.
   - **Version**: The version number of your Windows app. This should match the version as it appears in the `Package.appxmanifest` file.
4. Click the **Register application** button. From the main **Dashboard** page, your application is now listed under the default **mfp** runtime.

   Note: The `mfp` runtime is the default value for the **`wlServerContext`** parameter in the `mfpclient.resw` file below.
5. Click the application name to display the main configuration page for your app. The main page for your app displays different configuration options for the server-side registration of your app.



6. Click the **Configuration Files** tab. The Client Configuration File tab displays a template for creating your `mfpclient.resw` file. This file is used for connecting the client app to the server. For information on populating the values in the file and copying it to your Visual Studio project, see "Client property file for Windows 10 Universal Windows Platform and Windows 8 Universal."

Results

The app is registered on the target server.

What to do next

To complete the client-server registration, you must complete the required properties in the `mfpclient.resw` file and copy it to your Visual Studio project.

Client property file for Windows 10 Universal Windows Platform and Windows 8 Universal

This file defines the properties that your native app for Windows 10 Universal Windows Platform (UWP) and Windows 8 Universal requires to use the MobileFirst native API.

The `mfpclient.resw` client property file contains the necessary data to use the MobileFirst API for Windows 8 Universal. This file is added to your Visual Studio project, when you add the MobileFirst NuGet package. For more information, see "Adding the MobileFirst SDK by using NuGet" on page 7-66.

You must define the properties of this client property file before you use it in your native app for Windows 8 Universal.

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

Table 7-13. Properties and values of the `mfpclient.resw` file.

Property	Description	Example values
`wlServerProtocol`	The communication protocol with the MobileFirst Server. The **`wlServerProtocol`** property value can be `https` or `http`. In production, use `https`.	`http` or `https`.

Table 7-13. Properties and values of the `mfpclient.resw` file (continued).

Property	Description	Example values
`wlServerHost`	The public URL of the MobileFirst Server or of the cluster of MobileFirst Server. The host name must be accessible from the mobile devices. Do not use `localhost`, even for development tests, or your device or device simulators cannot reach the server.	`server.mycompany.com` or for tests an IP address, for example `169.254.184.88`.
`wlServerPort`	The port of the MobileFirst Server, for the protocol specified in **`wlServerProtocol`**.	`9080` for the HTTP port of the MobileFirst Development Server.
`wlServerContext`	The context root of the runtime component of MobileFirst Server that the app is registered to. The context root of the default runtime component is `/mfp/`.	`/mfp/`
`wlAppId`	The Package Identity Name, as registered in the MobileFirst Operations Console.	`Myapp.WindowsPhone`

Note: The client property file is placed in the following path: `strings/mfpclient.resw`.

When you register your app to a server with **`mfpdev app register`**, the client property file is automatically updated. For more information about registering the app with **`mfpdev`**, see “Registering Windows applications from the MobileFirst Platform CLI” on page 7-68.

You can also update the client property file with **`mfpdev app config`**. For more information, type **`mfpdev help app config`**.

Developing web applications

Use the IBM MobileFirst Platform Foundation web SDK to develop web applications with MobileFirst capabilities, including security features and application management.

Overview

You can develop mobile or Desktop MobileFirst web applications by using your preferred development environment and tools. To add MobileFirst features and capabilities to your client application, add the core module of the MobileFirst web SDK (`ibmmfpf.js`), which provides access to the JavaScript client-side API for development of web and cross-platform Cordova applications. To add MobileFirst Analytics capabilities to your application, also add the analytics module of the web SDK (`ibmmfpfanalytics.js`), which provides access to the JavaScript web analytics client-side API. You can also use the provided GUI and CLI tools to configure, manage, and secure your application.

Development steps

Follow the outlined procedure to develop your web application:

1. Select your preferred development tools and topology for developing the application. Ensure that your selected topology satisfies the requirements of the same-origin policy and “Google Chrome secure-origins policy” on page 7-75.

2. Download the sample MobileFirst starter web application (`MFPStarterWeb`), and use it either as the basis for your application or as a general reference for a functioning application. See "Getting started with a sample MobileFirst application" on page 7-25. The sample application already includes both the core and analytics modules of the web SDK. This step is optional.
3. Get the MobileFirst web SDK. See "Acquiring the MobileFirst web SDK" on page 7-75.
4. Add the required SDK files to your web application. See "Adding the MobileFirst SDK to web applications" on page 7-76. You can add the files to an existing web application, or create a new application. The sample `MFPStarterWeb` application already includes both the core and analytics modules of the web SDK, and demonstrates how to initialize the SDK and use its APIs.
5. Add JavaScript code to initialize the web SDK. See "Initializing the MobileFirst SDK" on page 7-79.
6. Develop your application. Use the JavaScript SDK APIs to add the required features and functions to your application.
7. Register your application to an instance of MobileFirst Server. See "Registering web applications to MobileFirst Server" on page 7-80.
8. Continue developing and testing your application.

   Note: To use server-side features, such as adapters and security, you must first register your application with a running instance of MobileFirst Server (see Step 7). For more information about the server and its installation, see "MobileFirst Server" on page 7-4 and "Setting up the development environment" on page 7-9.

Same-origin policy

You are free to host the web resources of your application on your preferred web server. However, when you select the development and production topologies, you must consider the restrictions of the same-origin-policy security model, which is designed to protect against potential security threats from unverified sources. According to this policy, a browser allows web resources (such as scripts) to interact only with resources that stem from the same origin (which is defined as a combination of URI scheme, host name, and port number). For more information about the same-origin policy, see The Web Origin Concept specification, and specifically 3. Principles of the Same-Origin Policy.

Because both your web server and MobileFirst Server need to communicate with your application, both servers must have the same web origin to satisfy the same-origin policy. Use one of the following methods to ensure a single web origin for your application:

Shared application server

Host your web resources on the same WebSphere Application Server as your MobileFirst Server runtime.

To implement this method, create a Maven webapp project (using the `maven-archetype-webapp` archetype), and build a web application archive (`.war` file) that contains your application's web resources. For information about creating Maven webapp projects, see Creating a webapp. Then, add your Maven web application to the WebSphere Application Server that hosts your MobileFirst Server, by editing the application server's configuration file (`server.xml`). For detailed step-by-step instructions, see Using WebSphere Liberty profile to serve the web application resources.

Reverse proxy

Set up a reverse proxy between your client application and its servers, and implement the proxy to redirect application requests to MobileFirst Server. The proxy acts as a single origin for all interaction with the application's web browser.

The sample MobileFirst starter web application (`MFPStarterWeb`) includes a reverse-proxy Node.js server that you can install with the Node Package Manager (**`npm`**), as outlined in the sample's `README.md` file.

For detailed information on how to create a custom Node.js server that acts as a single origin for your MobileFirst web application, see Using Node.js.

Google Chrome secure-origins policy

In production, it is recommended that you use the HTTPS protocol (HTTP over SSL) for the network communication with your web server and MobileFirst Server. However, during the development process you might prefer to use non-secure HTTP communication. In Google Chrome, HTTP communication with a remote server (not localhost) might cause an error due to the Chrome secure-origins policy. For more information about this policy, see Prefer Secure Origins For Powerful New Features. You can overcome this error by starting Chrome with the `--unsafely-treat-insecure-origin-as-secure` flag set to the IP address of your HTTP server. You also need to set the `--user-data-dir` flag to a profile-session directory. The following example overrides the security-origins policy for IP address `http://9.148.225.123:3000`, and uses a `/tmp/profle` for the browser session profile:

```
--args --unsafely-treat-insecure-origin-as-secure=http://9.148.225.123:3000 --user-data-dir=/tmp/profile
```

Acquiring the MobileFirst web SDK

Get a copy of the IBM MobileFirst Foundation web SDK for development of MobileFirst web applications.

Procedure

You can get the MobileFirst web SDK by using either of the following methods:

- Download the SDK from the Node Package Manager (npm) public software repository. The ibm-mfp-web-sdk npm package page displays the `README.md` file of the SDK package.

  Note: To download npm packages, you must first install the Node.js JavaScript runtime. For information about installing Node.js, see the Node js web site. From the command line, run the following command to download the SDK:

  ```
  npm install ibm-mfp-web-sdk
  ```

  The command creates, in the current directory, a `node_modules` directory that contains the `ibm-mfp-web-sdk` SDK directory. Consider running the command from the root directory of your web application to avoid manually copying the SDK directory when you add the SDK files to the application.
- Download the SDK from the Download Center of the IBM MobileFirst Platform Operations Console. This method does not require an internet connection. The method is available when the console is run on the MobileFirst Development Server from the IBM MobileFirst Platform Foundation Developer Kit. For detailed instructions, see “Acquiring the MobileFirst SDK from the MobileFirst Operations Console” on page 7-26. After you extract the downloaded archive file, you have a copy of the `ibm-mfp-web-sdk` SDK directory.

Results

You now have an SDK directory (`ibm-mfp-web-sdk`) with the required SDK files for developing a client MobileFirst web application.

What to do next

Add the required SDK files to an existing or new web application to add IBM MobileFirst Foundation capabilities and features, such as security, to the application. For an overview of the SDK modules and instructions for adding the modules to your application, see "Adding the MobileFirst SDK to web applications."

You can update your version of the MobileFirst web SDK, at any time, by running the following command from the root directory of your web application: `npm update ibm-mfp-web-sdk`. Remember that using npm requires Node js.

Adding the MobileFirst SDK to web applications

Add the MobileFirst web SDK to an existing or new web application to add IBM MobileFirst Foundation capabilities to the application.

Before you begin

Get a copy of the MobileFirst web SDK, and save the `ibm-mfp-web-sdk` SDK directory, or the containing `npm_modules` directory (if you downloaded the SDK from npm), to the root directory of your web application. For information about acquiring a copy of the web SDK, see "Acquiring the MobileFirst web SDK" on page 7-75.

About this task

To use the APIs that are exported by the MobileFirst JavaScript web-SDK modules, you must first load the appropriate modules into your web application:

- `ibmmfpf.js` - the core web-SDK module, which exports the JavaScript client-side API. You must load this module to develop a MobileFirst web application. This module enables you, for example, to add MobileFirst security capabilities to your application.
- `ibmmfpfanalytics.js` (provided in the `lib/analytics` directory) - the web-analytics module, which exports the MobileFirst JavaScript web analytics client-side API. Use this module to add IBM MobileFirst Analytics capabilities to your web application. For information about MobileFirst Analytics, see "Analytics and Logger" on page 11-1.

Note: `ibmmfpfanalytics.js` depends on `ibmmfpf.js`, and must be loaded first (when used).

The MobileFirst web SDK supports the common standards for loading JavaScript modules:

- In the Asynchronous Module Definition (AMD) standard, JavaScript modules are loaded asynchronously from JavaScript code. You use a module loader, such as RequireJS, to define your asynchronous modules and their dependencies. Then, you load the modules from the locations within your JavaScript code where the modules are needed, without affecting the global namespace. This method is especially suited for browser environments, and is commonly used in development of client web applications.

The sample MobileFirst starter web application (MFPStarterWeb), and the web-application samples that are available from the Developer Center, demonstrate how to use RequireJS to load the modules of the MobileFirst web SDK asynchronously. For more information about the starter sample and how to obtain a copy of the sample, see “Getting started with a sample MobileFirst application” on page 7-25.

- In the CommonJs Modules standard, JavaScript modules are loaded synchronously by importing the module into the global namespace. In web applications that use this standard, you import the modules from the <head> element of your main HTML file. This method is more suited for server environments, and is commonly used in development of Node.js servers.

Use one of the following alternative procedures, which correspond to the described load methods, to load the web-SDK modules:
- Asynchronous loading with RequiredJS
- Synchronous loading with HTML import

Note: The examples in the procedure assume that your SDK directory is found in the root of your application directory. If you select to store it in a different location, adjust the script paths in the examples. For example, if you downloaded the SDK with npm and saved the `node_modules` directory that contains the SDK directory in the root directory of your application, add `node_modules/` before each SDK module name.

Procedure

- **Asynchronous loading with RequiredJS**
  1. Get the RequireJS JavaScript module loader. For detailed instructions, see the RequireJS documentation.

     Note: The code examples in the following steps assume that `require.js` and your application's main HTML and JavaScript files are all stored in the application's root directory, which is used as the RequireJs `baseUrl`. You can select to save the scripts in different locations, and adjust the examples. For more information, see the RequireJS documentation.
  2. Load RequireJS by adding a <script> tag in the <head> tag of the application's main HTML file (typically `index.html`). Set the <script> tag's **`type`** attribute to `text/javascript`; set the **`src`** attribute to `require.js`; and set the **`data-main`** attribute to the path to your application's main JavaScript file (without the `.js` file extension) to instruct RequireJs to load the specified script after `require.js` loads. For more information, including alternative methods for adding `require.js`, see the RequireJS documentation. The following example is for an application with an `index.js` JavaScript file that is stored in the application's root directory together with the `reuiqre.js` script:
     ```
     <html>
         <head>
             <script type="text/javascript" src="require.js" data-main="index"></script>
         </head>
     </html>
     ```
  3. Use the methods of the `require` object that is created by RequireJS to load your application scripts from your application's JavaScript code.
     There are different ways for loading your scripts with RequireJS. This step demonstrates how to use the require.config method to configure module IDs for the SDK JavaScript modules; and then use the RequireJS require function

to load the module scripts by using the configured IDs instead of specifying the script paths. For more information and for alternative methods, see the RequireJS documentation.

a. In your application's main JavaScript file (for example, `index.js`), use the config method of the `require` object that is created by RequireJs to configure the SDK modules to load. The configuration specifies the paths to the script modules, and assigns each module an ID that is later used to load its script. The following example assigns the module ID `ibmmfpf` to the `ibmmfpf.js` module, and the module ID `ibmmfpfanalytics` to the `ibmmfpfanalytics.js` module.

```
require.config({
    'paths': {
        'ibmmfpfanalytics': 'ibm-mfp-web-sdk/lib/analytics/ibmmfpfanalytics',
        'ibmmfpf': 'ibm-mfp-web-sdk/ibmmfpf'
}
});
```

Note:

– If you select to use `ibmmfpfanalytics.js`, you must add it before `ibmmfpf.js` in the configuration paths.

– The names of the configured script modules are specified without the `.js` extension.

b. Use the RequireJS require function to load the SDK modules that you configured, from within your application's JavaScript code. The modules are loaded asynchronously into the local namespace when they are needed. The following example uses the `ibmmfpf` and `ibmmfpfanalytics` module IDs that you configured in the previous step to load the `ibmmfpf.js` and `ibmmfpfanalytics.js` SDK modules:

```
require(['ibmmfpfanalytics','ibmmfpf'], function(wlanalytics, WL)
```

Note: If you select to use `ibmmfpfanalytics.js`, you must load it before `ibmmfpf.js`.

- **Synchronous loading with HTML import** Load (import) the required modules from the <head> tag of the application's main HTML file (typically, `index.html`) by adding a <script> tag for each module. Set the script element's **`type`** attribute to `text/javascript`, and set its **`src`** attribute to the name of the target module. The following example loads both the core and analytics modules of the web SDK:

```
<html>
    <head>
        ...
        <script src="ibm-mfp-web-sdk/lib/analytics/ibmmfpfanalytics.js"></script>
        <script src="ibm-mfp-web-sdk/ibmmfpf.js"></script>
    </head>
</html>
```

Note: If you select to use `ibmmfpfanalytics.js`, you must load it before `ibmmfpf.js`.

What to do next

Use the initialization method of the core web API to initialize the SDK, and then develop your application by using the web SDK APIs. For more information, see "Initializing the MobileFirst SDK" on page 7-79.

Initializing the MobileFirst SDK

Initialize the MobileFirst web SDK before using its APIs to develop your web application.

Before you begin

Download the MobileFirst web SDK and add it to your web application. For more information, see “Adding the MobileFirst SDK to web applications” on page 7-76.

About this task

Add the initialization code that is described in the following procedure to your application's JavaScript code (for example, in `index.js`).

Procedure

1. Define an initialization-properties JSON object that sets the following mandatory SDK initialization properties:
   - **`mfpContextRoot`** - the context root of your application's MobileFirst Server runtime.
   - **`applicationId`** - a unique identifier for your application.

     Note: The application ID that is set in this initialization option must be the same ID that you provide when your register the application. For more information, see “Registering web applications to MobileFirst Server” on page 7-80.

   For example, the following code defines an *mfpInitProperties* variable that sets the context root of MobileFirst Server (**`mfpContextRoot`**) to `/mfp` (the default context root of the MobileFirst Development Server), and sets the application ID (**`applicationId`**) to `com.example.myapplication`:

   ```
   var mfpInitProperties = {
       'mfpContextRoot' : '/mfp' ,
       'applicationId' : 'com.example.myapplication'
   };
   ```
2. Call the WL.Client.init method of the core web-SDK module (`ibmmfpf.js`) to initialize the SDK. Pass the initialization-properties JSON object that you defined in the previous step (*mfpInitProperties*) as the parameter of this method. Use a JavaScript promise to implement the initialization-completion logic of the asynchronous init method, as demonstrated in the following example:

   ```
   WL.Client.init(mfpInitProperties).then
       (function(){
           console.log('MobileFirst web SDK initialized');
           // Application initialization logic
   });
   ```

   Note: For backwards compatibility with Cordova applications that use the same MobileFirst client JavaScript API, you can select to implement the initialization-completion logic in a wlCommonInit function instead of using a JavaScript promise.

What to do next

You can now use the JavaScript client-side API to develop your application and add MobileFirst capabilities. If you selected to add the web-analytics module of the SDK (`ibmmfpfanalytics.js`), as outlined in “Adding the MobileFirst SDK to web applications" on page 7-76, use the JavaScript web analytics client-side API to add MobileFirst Analytics features to your application (see "Analytics and Logger" on page 11-1).

To use server-side features, such as adapters and security, first register your application to a running instance of MobileFirst Server. See "Registering web applications to MobileFirst Server."

Registering web applications to MobileFirst Server

Register your web application to an instance of MobileFirst Server to establish communication with the server and provide the server with information about your application. You can register the application by using either the IBM MobileFirst Platform Command Line Interface (CLI) or the IBM MobileFirst Platform Operations Console.

Registering web applications from the MobileFirst Platform CLI:

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to register your web application to an instance MobileFirst Server.

Before you begin

- Install the IBM MobileFirst Platform Command Line Interface (CLI). For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- Set up a running instance of MobileFirst Server. The server can run locally or it can be a remote server, but you must be able to connect to the server from your local development machine. For more information, see "Setting up the MobileFirst Development Server" on page 7-12, "Installing the IBM MobileFirst Platform Foundation Developer Kit" on page 7-10, or "Installing IBM MobileFirst Platform Server" on page 6-2.

Procedure

From the command line, run one of the following MobileFirst Platform CLI command variations to register your application to an instance of MobileFirst Server:

- To register the application to the default MobileFirst Server, run the following command, where <*applicationId*> is a unique identifier for your application:
  ```
  mfpdev app register <applicationId>
  ```
  For example, the following command registers an application with the application ID `com.example.myapplication` to the default MobileFirst Server:
  ```
  mfpdev app register com.example.myapplication
  ```
  Note: If you did not previously set a default server, and an instance of MobileFirst Server is running on your local system, the command registers the application to the running local server and this server becomes the default MobileFirst Server.
- To register your application to a server instance other than the default MobileFirst Server, follow these steps:
  1. Run the **`mfpdev server add`** command to create your preferred server profile. For example:

```
mfpdev server add Server1 -url https://company.mobile.com:9080 -login admin -password secretPassword!
```

For more information about the **`mfpdev server add command`**, run `mfpdev help server add`.

2. Run the following command to register your application to the server that you defined in the previous step, where *<applicationId>* is a unique identifier for your application and *<server>* is the name of the server profile:

   ```
   mfpdev app register <applicationId> <server>
   ```

   Note: The application ID that is provided during registration must be identical to the value of the **`applicationId`** initialization option that is set when initializing the web SDK. For more information, see "Initializing the MobileFirst SDK" on page 7-79.
   For example, the following command registers an application with the application ID `com.example.myapplication` to a `Server1` MobileFirst Server:

   ```
   mfpdev app register com.example.myapplication Server1
   ```

   For more information about the **`mfpdev app register`** command, including optional parameters, run **`mfpdev help app register`**.

Results

When the application is successfully registered to the target server, you can select it from the **Applications** section in the navigation sidebar of the IBM MobileFirst Platform Operations Console (under **Versions** or **Web**). The main application page displays different server-side configuration options for your application.

In the application **Configuration Files** console tab, you can see the content of the application's descriptor and runtime configuration files. The value of the **`applicationId`** in the application descriptor is the application ID that you specified when you registered the application.

Note: You cannot have multiple versions of the same web application. Therefore, the `applicationKey` object of the application descriptor does not contain a `version` property.

What to do next

You can proceed with other development tasks that depend on MobileFirst Server, such as making server-side configurations, testing the application's security features, and managing the application from the MobileFirst Operations Console.

Make sure to initialize the SDK from your application before calling any SDK APIs other than the initialization method. For more information, see "Initializing the MobileFirst SDK" on page 7-79.

Registering web applications from the MobileFirst Operations Console:

You can use the IBM MobileFirst Platform Operations Console to register your web application to an instance of MobileFirst Server.

Before you begin

Run the IBM MobileFirst Platform Operations Console on the same instance of MobileFirst Server to which you want to register your application. For information about how to install the MobileFirst Development Server and run the console, see "Installing the IBM MobileFirst Platform Foundation Developer Kit" on page 7-10.

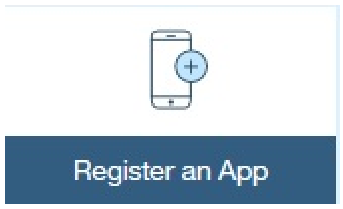
Note: You can register an application to the server either before or after you set up your development environment and start developing the application. However, you must register your application before you can call the MobileFirst APIs from your client application and communicate with the server.

Procedure

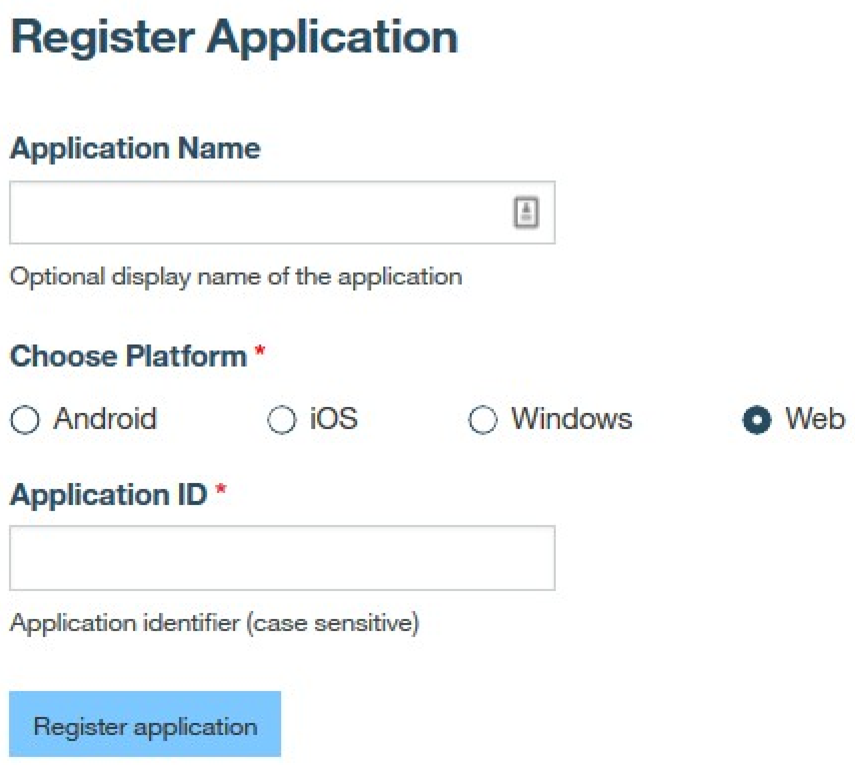
1. In the MobileFirst Operations Console, navigate to the Register Application page by using one of the following methods:
   - In the navigation sidebar, select **New** next to **Appilcations**.

     Applications (0)
New

   - From the **Dashboard**, select **Register an App**.

2. On the Register Application page, provide the required input for registering your web application, as demonstrated in the following image.

   Register Application

   Application Name

   Optional display name of the application

   Choose Platform *

   Android iOS Windows [✓ Web]

   Application ID *

   Application identifier (case sensitive)

   a. In the **Application Name** field, optionally enter a display name for your application (for example, `MyApplication`). If you do not provide an application name, the application ID is used also as the display name.
   b. In the **Choose Platform** field, select **Web**.

c. In the **Application ID** field, enter a unique identifier for your application (for example, `com.example.myapplication`).

   Note: The application ID that is provided during registration must be identical to the value of the **`applicationId`** initialization option that is set when initializing the web SDK. For more information, see "Initializing the MobileFirst SDK" on page 7-79.

3. Select **Register application** to complete the registration.

Results

When the application is successfully registered to the target server, you can select it from the **Applications** section in the navigation sidebar of the MobileFirst Operations Console (under **Versions** or **Web**). The main application page displays different server-side configuration options for your application.

In the application **Configuration Files** console tab, you can see the content of the application's descriptor and runtime configuration files. The value of the **`applicationId`** in the application descriptor is the application ID that you specified when you registered the application.

Note: You cannot have multiple versions of the same web application. Therefore, the `applicationKey` object of the application descriptor does not contain a `version` property.

What to do next

You can proceed with other development tasks that depend on MobileFirst Server, such as making server-side configurations, testing the application's security features, and managing the application from the MobileFirst Operations Console.

Make sure to initialize the SDK from your application before calling any SDK APIs other than the initialization method. For more information, see "Initializing the MobileFirst SDK" on page 7-79.

Developing Cordova applications

To develop cross-platform MobileFirst applications, you use the open-source Apache Cordova tooling and framework. Certain MobileFirst features specifically support integration with Cordova apps.

Apache Cordova provides a web-based platform for developing apps using HTML, CSS, and JavaScript within the application WebView interface. The WebView is started by the native SDK. MobileFirst SDKs provide functionality for both stages of the runtime, with a minimal number of necessary functions built on the native MobileFirst SDK. In Cordova apps, you can write JavaScript code by using standard web technologies such as CSS, HTML, and JavaScript, without requiring recoding in each native mobile platform's development language. Instead, applications execute within native wrappers targeted to each native platform. The Crosswalk WebView, which can replace the native Cordova WebView, is newly supported by the MobileFirst SDK in V8.0.0. . For more information, see "Crosswalk WebView (Android)" on page 7-133.

To create Cordova apps that are enabled for MobileFirst features, you use your preferred development tools that support Cordova such as the Apache Cordova CLI, Ionic, or IntelliJ. You obtain some software, such as plug-ins, code to support various target platforms, command-line tools, or an IDE directly from the supplier of your development tools. Then, to use MobileFirst capabilities, such as back-end services of MobileFirst Server, you add Cordova plug-ins that support MobileFirst features to your app. These plug-ins are included in the IBM MobileFirst Platform Foundation Developer Kit. You can also download these plug-ins from npm or JazzHub. For information about creating the Cordova app and adding MobileFirst Server functionality, see "Creating Cordova apps that include MobileFirst features" on page 7-87.

Important: Cordova development tools are not provided with IBM MobileFirst Platform Foundation.

After installing all necessary plug-ins, register your app on the server. For more information, see "Registering Cordova applications to MobileFirst Server" on page 7-106.

Develop your own application code in your chosen IDE and view it by using built-in emulators in Xcode, Android Studio, and Cordova (`cordova emulate`).

- For information about development issues specific to the native environment, changes made to the startup code by the MobileFirst plug-in, details on starting the Cordova WebView, see "Developing Cordova apps for Android" on page 7-121, "Developing Cordova apps for iOS" on page 7-124, and "Developing Cordova apps for Windows" on page 7-128.
- For information about developing the WebView and IDE editors providing autocomplete for the JavaScript API, see "Cordova WebView" on page 7-129.

The following topics provide details for each stage of development.

Prerequisites for developing Cordova apps with MobileFirst features

To develop Cordova apps that include MobileFirst functionality, the following software is required.

Requirements of Apache Cordova

You must have the software that is required by Apache Cordova for all of your target platforms. For information about those requirements, refer to the platform requirements documentation from Apache Cordova. For example:

- Android: https://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html
- Windows: https://cordova.apache.org/docs/en/latest/guide/platforms/win8/index.html
- iOS: https://cordova.apache.org/docs/en/latest/guide/platforms/ios/index.html

For more information, see the Apache Cordova documentation.

Node.js and Node Package Manager (npm)

Node Package Manager, or npm, is a public software repository. The Apache Cordova command-line interface (CLI), the MobileFirst Platform CLI, and the plug-ins that contain the MobileFirst SDK and other MobileFirst features are hosted on JazzHub and npm.

You must install Node.js version 4.0.0 or later to be able to download and run packages from npm. For information about installing Node.js, see the Node js web site. For information about installing npm, see How to install npm.

Note: You must install Node.js to obtain the Apache Cordova CLI from npm. You can install the MobileFirst Platform CLI and the plug-ins that contain the MobileFirst SDK and other MobileFirst features from npm or JazzHub, but alternatively, they are provided with the IBM MobileFirst Platform Foundation Developer Kit.

Cordova command-line interface (CLI)

To develop Cordova applications with MobileFirst features, you must have the Apache Cordova command-line interface installed. You must have version 6.1.1.

You might want to use other development tools that support Apache Cordova as part of your development environment. If so, they usually also require that you install the Apache Cordova CLI. Make sure that your preferred Cordova development tools support Cordova 6.1.1.

For information about installing the Cordova CLI, see Installing the Cordova CLI at the Apache Cordova website.

MobileFirst components

For your MobileFirst development, you can choose whether to install the IBM MobileFirst Platform Foundation Developer Kit or to separately install various MobileFirst components.

IBM MobileFirst Platform Foundation Developer Kit

The IBM MobileFirst Platform Foundation Developer Kit contains, in one package, all of the components of IBM MobileFirst Platform Foundation that are needed to start developing your MobileFirst app.

If you use the IBM MobileFirst Platform Foundation Developer Kit, you don't need additional MobileFirst components until you reach the stage in your development process when you need the full features of MobileFirst Server rather than the MobileFirst Development Server that is provided with the IBM MobileFirst Platform Foundation Developer Kit. At some point, you might also want to download the latest versions of MobileFirst plug-ins that are available for download from the JazzHub repository or npm.

For more information about the IBM MobileFirst Platform Foundation Developer Kit, see “The IBM MobileFirst Platform Foundation Developer Kit” on page 7-9.

Separate MobileFirst components

If you choose to install separate components, you need the following elements:

IBM MobileFirst Platform Server

You can use either this stand-alone MobileFirst Server that is provided with IBM MobileFirst Platform Foundation or the MobileFirst Development Server that is provided with the IBM

MobileFirst Platform Foundation Developer Kit. For more information about the stand-alone MobileFirst Server, see "MobileFirst Server" on page 7-4.

IBM MobileFirst Platform Command Line Interface (CLI)

The MobileFirst Platform CLI includes certain commands that specifically support Cordova development.

You can download the latest version of the CLI with npm, or you can use the version that is provided with the IBM MobileFirst Platform Foundation Developer Kit. For more information about the CLI, see "The MobileFirst command-line interface (CLI)" on page 7-13.

Cordova plug-ins for enabling MobileFirst features in your app

To develop a Cordova application that includes IBM MobileFirst Platform Foundation features, you must add MobileFirst plug-ins to your app. The plug-ins are available in the IBM MobileFirst Platform Foundation Developer Kit, or you can download them from npm or JazzHub. For more information, see "Cordova plug-ins for MobileFirst features" on page 7-87.

Supported Cordova components for MobileFirst cross-platform apps

These are the versions of Apache Cordova components that have been tested with the current version of IBM MobileFirst Platform Foundation. Ensure that you are using these versions to avoid possible compatibility issues.

You add these Cordova components to your application by using your preferred Cordova development tools.

Platforms

- cordova-android: 5.1.1
- cordova-ios: 4.1.1
- cordova-windows: 4.3.2

Important: If you are updating your cordova-ios platform, you must use the `cordova rm platform ios` command to uninstall the platform and then install it using the `cordova platform add ios` command. The update fails if you use the `cordova platform update ios` command. See "Known limitations" on page 3-25 for more information.

Plug-ins

The following plug-ins are required by IBM MobileFirst Platform Foundation:

- cordova-plugin-device: 1.1.1
- cordova-plugin-dialogs: 1.2.0
- cordova-plugin-globalization: 1.0.2
- cordova-plugin-okhttp: 2.0.0

The following plug-ins are not required by IBM MobileFirst Platform Foundation but are likely to be required by your app:

- cordova-plugin-splashscreen: 3.2.0
- cordova-plugin-whitelist: 1.2.1

Tools

Cordova command line (Cordova CLI) : 6.1.1

Creating Cordova apps that include MobileFirst features

You can create a Cordova app that is enhanced with MobileFirst features in several ways.

In all cases, you use your preferred Cordova development tools such as the Apache Cordova command-line interface, the Ionic Framework, or IntelliJ to set up and work with your app. Then, you add the IBM MobileFirst Platform Foundation SDK and other MobileFirst features to your app in the form of Cordova plug-ins that support MobileFirst features.

You can accomplish this in any of the following ways:
- When you initially create your Cordova app, specify to include the MobileFirst template.
- Create a new Cordova app, and then add in the MobileFirst plug-ins.
- Add the MobileFirst plug-ins to an existing Cordova app.

Cordova plug-ins for MobileFirst features:

To add MobileFirst functionality to your app, you use Cordova plug-ins that contain MobileFirst features.

The Cordova plug-ins for IBM MobileFirst Platform Foundation provide MobileFirst functions such as support for adapters, security, monitoring, and server access for Cordova apps that were initially created with non-IBM MobileFirst Platform Foundation tools.

The plug-ins are included with the IBM MobileFirst Platform Foundation Developer Kit, or you can download the latest versions from npm and JazzHub. You download and install these hosted plug-ins by using third party tools, such as the Cordova CLI or Ionic Framework CLI.

The following plug-ins are available:
- `cordova-plugin-mfp`
- `cordova-plugin-mfp-push`
- `cordova-plugin-mfp-jsonstore`
- `cordova-plugin-mfp-fips`
- `cordova-plugin-mfp-encrypt-utils`

The `cordova-plugin-mfp` plug-in contains the core MobileFirst functions and is required. If you install either the `cordova-plugin-mfp-push` plug-in or the `cordova-plugin-mfp-jsonstore` plug-in, the `cordova-plugin-mfp` is automatically installed.

The `cordova-plugin-mfp-jsonstore` plug-in enables your app to use JSONstore. For more information on JSONstore, see "JSONStore overview" on page 7-134.

The `cordova-plugin-mfp-push` plug-in provides permissions needed to use push notification from the MobileFirst Server for Android and iOS apps. For more information about push notification, see "Push notification" on page 7-248.

The `cordova-plugin-mfp-fips` plug-in provides FIPS 140-2 support for Android platforms. For more information, see "FIPS 140-2 support" on page 10-75.

The `cordova-plugin-mfp-encrypt-utils` plug-in provides iOS OpenSSL frameworks for encryption for Cordova applications with the iOS platform. For more information, see "Enabling OpenSSL for Cordova iOS" on page 7-126.

Known limitation for Cordova application loaded with cordova-plugin-mfp: `cordova-plugin-statusbar` will not work with Cordova application loaded with `cordova-plugin-mfp`. For more information about the limitation and steps to circumvent it, see "cordova-plugin-statusbar does not work with Cordova application loaded with cordova-plugin-mfp." on page 3-32.

Creating a new Cordova app with the MobileFirst template:

You can use a template to easily create a simple Cordova app that is enabled for IBM MobileFirst Platform Foundation. You can use this app as a starting point to build your own.

Before you begin

- You must have Cordova development tools installed. This example uses the Apache Cordova CLI. If you use other Cordova development tools, some of your steps will be different. Refer to your Cordova tool documentation for instructions.
- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.

About this task

The template creates a simple mobile app that displays the text `Hello World`. To create your Cordova application with the template, you add the template when you initially create your Cordova app with your Cordova development tools.

The template automatically adds the following plug-ins to your app: `cordova-plugin-mfp`, `cordova-plugin-whitelist` and `cordova-plugin-splashscreen`. The `cordova-plugin-mfp` plug-in is required for a MobileFirst Cordova app. Most applications will need the `cordova-plugin-whitelist` and `cordova-plugin-splashscreen` plug-ins. If your app will not need these, you can remove them after the app is initially created.

The following steps show how to create your app with the template by using the Apache Cordova CLI:

Procedure

Enter the following command in your command window, where *AppName* is the name of the new app:

```
cordova create AppName  --template cordova-template-mfp
```

Results

In the `www` directory of your of your app, (*`AppName`*`/www`), the `index.html` file contains a simple app that displays `Hello World`.

What to do next

Add platforms, additional MobileFirst plug-ins, and optionally, other plug-ins to your app. For more information, see "Creating a new Cordova app without the MobileFirst template." If your app does not need the `cordova-plugin-whitelist` or `cordova-plugin-splashscreen` plug-ins, you can remove them.

To further develop your app with features that depend on the MobileFirst Server, register your app with the server. For more information, see "Registering Cordova applications from the MobileFirst Platform CLI" on page 7-107.

Important: (Android and iOS development only.) If you are developing your app for the Android or iOS platforms, when you add the platform to your app that contains the `cordova-plugin-mfp` plug-in, an existing file in your app is replaced by a version of the file that is provided by MobileFirst Platform Foundation. These files are:

- Android: The file `MainActivity.java` is replaced. Your original `MainActivity.java` file is backed up and renamed `MainActivity.original`.
- iOS: File `main.m` is replaced. Your original `main.m` file is backed up and renamed `main.m.bak`.

If you made any changes to the original versions of these files, you must merge the changes that you made into the new version of the file that is provided by IBM MobileFirst Platform Foundation.

Creating a new Cordova app without the MobileFirst template:

You can create a new Cordova app that contains MobileFirst features with a few simple commands.

Before you begin

- You must have Cordova development tools installed. This example uses the Apache Cordova CLI. If you use other Cordova development tools, some of your steps will be different. Refer to your Cordova tool documentation for instructions.
- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.

About this task

You enter all of the commands in the following steps from your command window.

Procedure

1. To initially create your Cordova application, enter `cordova create` *`AppName`* where *`AppName`* is the name of the application that you are creating. For example, the following command creates an application named MyApp:

   ```
   cordova create MyApp
   ```

   Important: You can optionally include a MobileFirst template when you initially create your app. In this case, some Cordova plug-ins are added for you automatically. For more information, see "Creating a new Cordova app with the MobileFirst template" on page 7-88.
2. Enter `cd` *`AppName`* to change to the root directory of your new app.
3. To add platforms to your app, enter `cordova platform add` *`platform`* where *`platform`* is `ios`, `windows`, or `android`, or any combination of these platforms.

   Important: Verify that the version of the platform that you add is supported by IBM MobileFirst Platform Foundation. For a list of minimum supported platform version levels, see "Supported Cordova components for MobileFirst cross-platform apps" on page 7-86. The latest available platform versions will be downloaded by default with the commands described in this step. To download a previous version of a platform, specify the version number by using the syntax `cordova platform add` *`platform@version`*. For example: `cordova platform add windows@4.3.0`.
   For example, the following command adds the iOS, Android, and Windows platforms:

   ```
   cordova platform add ios android windows
   ```
4. To add the IBM MobileFirst Platform Foundation SDK plug-in, enter `cordova plugin add cordova-plugin-mfp`.

   Note: If you used the MobileFirst template when you created your app, you do not have to complete this step because the `cordova-plugin-mfp` plug-in has already been added to your app.
5. To add additional Cordova plug-ins for MobileFirst, enter `cordova plugin add` *`plugin`* where *`plugin`* is one of the following:
   - `cordova-plugin-mfp-push`
   - `cordova-plugin-mfp-jsonstore`
   - `cordova-plugin-mfp-fips`
   - `cordova-plugin-mfp-encrypt-utils`

   For example, the following command installs MobileFirst FIPS 140-2 for Android and JSONStore support:

   ```
   cordova plugin add cordova-plugin-mfp-fips cordova-plugin-mfp-jsonstore
   ```
6. Optional: Add other Cordova plug-ins that might be used by your app. For more information see The Command-Line Interface ("Add Plugin Features" section) on the Apache Cordova web site.

   For example, to add the Cordova whitelist plug-in:

   ```
   cordova plugin add cordova-plugin-whitelist
   ```

   Or, for example, to add the CrossWalk plug-in:

   ```
   cordova plugin add cordova-plugin-crosswalk-webview --variable XWALK_VERSION="org.xwalk:xwalk_core_library:15+"
   ```

   Note: Some versions of CrossWalk can cause a problem when you run the app on an Android emulator. For more information, see "Known limitations" on page 3-25.

Results

You now have a Cordova app that is enabled for IBM MobileFirst Platform Foundation. In the case of the example used here, the Cordova application MyApp has the iOS and Android platforms installed, and also has FIPS 140-2 support for Android and JSONStore support (for all platforms) installed. The Cordova whitelist and CrossWalk plug-ins are optionally installed.

What to do next

To further develop your app with features that depend on the MobileFirst Server, register your app with the server. For more information, see "Registering Cordova applications from the MobileFirst Platform CLI" on page 7-107.

Important: (Android and iOS development only.) If you are developing your app for the Android or iOS platforms, when you add the platform to your app that contains the `cordova-plugin-mfp` plug-in, an existing file in your app is replaced by a version of the file that is provided by MobileFirst Platform Foundation. These files are:

- Android: The file `MainActivity.java` is replaced. Your original `MainActivity.java` file is backed up and renamed `MainActivity.original`.
- iOS: File `main.m` is replaced. Your original `main.m` file is backed up and renamed `main.m.bak`.

If you made any changes to the original versions of these files, you must merge the changes that you made into the new version of the file that is provided by IBM MobileFirst Platform Foundation.

Adding MobileFirst features to an existing Cordova app:

You can add capabilities provided by IBM MobileFirst Platform Foundation to an existing Cordova app that you created with Apache Cordova, Ionic, or other third-party tools by adding the IBM MobileFirst Platform Foundation SDK to your app. The SDK is provided in the form of Cordova plug-ins.

Before you begin

- You must already have a Cordova application on your system that you created with third-party tools such as the Apache Cordova command-line interface or the Ionic Framework command-line interface.
- Your existing Cordova app must already include the applicable Apache Cordova platforms. The version of a platform must be at least the version supported by IBM MobileFirst Platform Foundation. For a list of minimum supported platform version levels, see "Supported Cordova components for MobileFirst cross-platform apps" on page 7-86. If a platform in your app is at a lower version level than is in this list, upgrade to the version level on the list.
- Your existing Cordova app most likely includes some Apache Cordova plug-ins. The version of any Apache Cordova plug-in that is used by your app must be at least the version supported by IBM MobileFirst Platform Foundation. For a list of minimum supported plug-in version levels, see "Supported Cordova components for MobileFirst cross-platform apps" on page 7-86. If a plug-in in your app is at a lower version level than is in this list, upgrade to the version level on the list.

- You must have the Cordova Command-Line Interface (CLI) installed, and any prerequisites installed that are required for using the Cordova CLI for your target platforms. For more information, see The Command-Line Interface on the Apache Cordova web site.
- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have internet access.
- You must have node.js version 4.0.0 or later installed.

About this task

You can add any of the following plug-ins to your app.
- `cordova-plugin-mfp`
- `cordova-plugin-mfp-push`
- `cordova-plugin-mfp-jsonstore`
- `cordova-plugin-mfp-fips`
- `cordova-plugin-mfp-encrypt-utils`

The `cordova-plugin-mfp` provides the MobileFirst SDK and is required. See "Cordova plug-ins for MobileFirst features" on page 7-87 for information about the function of each of the other plug-ins.

Procedure

1. To review the readme file for the plug-ins that you want, navigate to the download page for each one on the npm web site in your web browser. Use one or more of the following URLs, depending on the plug-in:
   - https://www.npmjs.com/package/cordova-plugin-mfp
   - https://www.npmjs.com/package/cordova-plugin-mfp-push
   - https://www.npmjs.com/package/cordova-plugin-mfp-jsonstore
   - https://www.npmjs.com/package/cordova-plugin-mfp-fips
   - https://www.npmjs.com/package/cordova-plugin-mfp-encrypt-utils

   Note: You can also download the compressed (.zip) file packages from JazzHub. Click the link that starts with `hub.jazz.net` from the npm page for a given plug-in to reach the download page.
2. On your local system, navigate to the root folder of your Cordova app.
3. Run one of the following commands, depending on which plug-ins you want to install.

   Note: The following steps use the Cordova CLI. If you are using different Cordova-compatible tools, the procedure might differ.

   Important: Check your app's components for compatibility issues, to make sure that they will work with the latest plug-ins. The latest available versions of plug-ins will be downloaded by default with the commands described in this step. To download a specific version of a plug-in, specify the version number by using the syntax `cordova plugin add` *`plugin_name@version`*. For example, to download the latest published plug-in for IBM MobileFirst Platform Foundation 8.0, use `cordova plugin add cordova-plugin-mfp@8.0`. If you want to import a specific version such as 8.0.2016021411, replace with the version number you are using, including the major minor and patch numbers. The patch number is in the format *YYYYMMDDHH.*. For example: `cordova plugin add cordova-plugin-mfp@8.0.2016021411`.

IBM MobileFirst Platform Foundation

- To install the core MobileFirst Cordova plug-in:

  ```
  cordova plugin add cordova-plugin-mfp
  ```

  Or if you acquired the Cordova SDK from MobileFirst Operations Console

  ```
  cordova plugin add <unzip_dir>/plugins/cordova-plugin-mfp
  ```

  Where `<unzip_dir>` is the directory where you unzipped the acquired SDK.
- To install the `cordova-plugin-mfp-push` and `cordova-plugin-mfp` plug-ins:

  ```
  cordova plugin add cordova-plugin-mfp-push
  ```
- To install the `cordova-plugin-mfp-jsonstore` and `cordova-plugin-mfp` plug-ins:

  ```
  cordova plugin add cordova-plugin-mfp-jsonstore
  ```
- To install the `cordova-plugin-mfp-fips` and `cordova-plugin-mfp` plug-ins:

  ```
  cordova plugin add cordova-plugin-mfp-fips
  ```
- To install the `cordova-plugin-mfp-encrypt-utils` and `cordova-plugin-mfp` plug-ins:

  ```
  cordova plugin add cordova-plugin-mfp-encrypt-utils
  ```

Note: If you install `cordova-plugin-mfp` by itself, you can add the `cordova-plugin-mfp-push` , `cordova-plugin-mfp-jsonstore` , `cordova-plugin-mfp-fips` or `cordova-plugin-mfp-encrypt-utils` plug-ins later by specifying them with the **`cordova plugin add`** command.

Results

The IBM MobileFirst Platform Foundation SDK plug-ins for Cordova are installed.

What to do next

You can now start developing the IBM MobileFirst Platform Foundation capabilities in your Cordova app.

To further develop your app with features that depend on the MobileFirst Server, register your app with the server. For more information, see "Registering Cordova applications from the MobileFirst Platform CLI" on page 7-107.

Important: (Android and iOS development only.) If you are developing your app for the Android or iOS platforms, when you add the platform to your app that contains the `cordova-plugin-mfp` plug-in, an existing file in your app is replaced by a version of the file that is provided by MobileFirst Platform Foundation. These files are:

- Android: The file `MainActivity.java` is replaced. Your original `MainActivity.java` file is backed up and renamed `MainActivity.original`.
- iOS: File `main.m` is replaced. Your original `main.m` file is backed up and renamed `main.m.bak`.

If you made any changes to the original versions of these files, you must merge the changes that you made into the new version of the file that is provided by IBM MobileFirst Platform Foundation.

Inside your Cordova app

Some aspects of your Cordova app are changed when you enable it for IBM MobileFirst Platform Foundation.

The Cordova configuration file, `config.xml`, now includes MobileFirst information and resources such as splash screens and icons are provided when you use the MobileFirst template.

The Cordova configuration file:

The Cordova `config.xml` file is the global configuration file of the application.

The Cordova configuration file is a mandatory XML file that contains application metadata, and is stored in the root directory of the app. The file is automatically generated when you create a Cordova application. You can then modify it to add custom properties.

You can find more general information about this file in the Apache Cordova config.xml documentation.

To know more about any MobileFirst-specific configuration, refer to the following sections.

Structure of the `config.xml` file

To view detailed information about any MobileFirst-specific configuration, click the element name in the file description, or in the alphabetical list that follows the description.

```
<?xml version='1.0'encoding='utf-8'?>
<widget>
   <name>...</name>
   <description>...</description>
   <author>...</author>
   <content/>
   <plugin/>
   <access/>
   <allow-intent/>
   <platform>
      <allow-intent/>
      <icon/>
      <splash/>
      <update/>
   </platform>
   <mfp:platformVersion>...</mfp:platformVersion>
   <mfp:directUpdateAuthenticityPublicKey>...</mfp:directUpdateAuthenticityPublicKey>
   <mfp:languagePreferences>...</mfp:languagePreferences>
   <mfp:clientCustomInit/>
   <mfp:server/>
   <mfp:ios>
      <mfp:appChecksum>...</mfp:appChecksum>
      <mfp:sdkChecksum>...</mfp:sdkChecksum>
      <mfp:security>
         <mfp:testWebResourcesChecksum/>
      </mfp:security>
   </mfp:ios>
   <mfp:android>
      <mfp:appChecksum>...</mfp:appChecksum>
      <mfp:sdkChecksum>...</mfp:sdkChecksum>
      <mfp:security>
         <mfp:testWebResourcesChecksum/>
      </mfp:security>
   </mfp:android>
   <mfp:windows>
      <mfp:appChecksum>...</mfp:appChecksum>
      <mfp:windowsphone8>
         <mfp:sdkChecksum>...</mfp:sdkChecksum>
         <mfp:security>
            <mfp:testWebResourcesChecksum/>
         </mfp:security>
      </mfp:windowsphone8>
      <mfp:windows8>
         <mfp:sdkChecksum>...</mfp:sdkChecksum>
         <mfp:security>
            <mfp:testWebResourcesChecksum/>
         </mfp:security>
      </mfp:windows8>
      <mfp:windows10>
         <mfp:sdkChecksum>...</mfp:sdkChecksum>
         <mfp:security>
            <mfp:testWebResourcesChecksum/>
         </mfp:security>
      </mfp:windows10>
   </mfp:windows>
</widget>
```

`config.xml` elements

- `<mfp:android>`
- `<mfp:appChecksum>`
- `<mfp:clientCustomInit/>`
- `<mfp:directUpdateAuthenticityPublicKey>`
- `<mfp:ios>`
- `<mfp:languagePreferences>`
- `<mfp:platformVersion>`
- `<mfp:sdkChecksum>`
- `<mfp:security>`
- `<mfp:server/>`
- `<mfp:testWebResourcesChecksum/>`
- `<mfp:windows>`
- `<mfp:windows8>`
- `<mfp:windowsphone8>`
- `<mfp:windows10>`
- `<widget>`

`<widget>`

Syntax

```
<widget id="my.mfp.cordova.with.push.plugin"
        version="1.0.5"
        xmlns="http://www.w3.org/ns/widgets"
        xmlns:cdv="http://cordova.apache.org/ns/1.0"
        xmlns:mfp="http://www.ibm.com/mobilefirst/cordova-plugin-mfp">
   ...
</widget>
```

Description

See the Apache Cordova config.xml documentation.

MobileFirst-specific attributes

- `id`: This is the application package name that was specified when the Cordova project was created. If this value is manually changed after the application was registered with the MobileFirst Server, then the application must be registered again.
- `xmlns:mfp`: Required. Set by default. It is the MobileFirst plug-in XML namespace.

Back to top

`<mfp:platformVersion>`

Syntax

```
<mfp:platformVersion>8.0.0.00.20160205-2039</mfp:platformVersion>
```

Contained in
`<widget>`

Configuration
Set by default. Must not be changed.

Description
Required. The product version on which the application was developed.

Back to top

`<mfp:directUpdateAuthenticityPublicKey>`

Syntax

```
<mfp:directUpdateAuthenticityPublicKey>public_key_string</mfp:directUpdateAuthenticityPublicKey>
```

Contained in
`<widget>`

Configuration
Set with the **`mfpdev app config direct_update_authenticity_public_key <value>`** command.

For more details about the**`mfpdev app config`** command, type **`mfpdev help app config`** in your command window.

Description
Optional. When you enable the Direct Update Authenticity feature, the direct update package is digitally signed during deployment. After the client downloaded the package, a security check is run to validate the package authenticity. This string value is the public key that will be used to authenticate the direct update `.zip` file. For more information, see "Implementing secure Direct Update on the client side" on page 7-239.

Note: This element value is only supported by the platforms that support direct update: Cordova iOS and Cordova Android.

Back to top

`<mfp:languagePreferences>`

Syntax

```
<mfp:languagePreferences>en</mfp:languagePreferences>
```

Contained in
`<widget>`

Configuration
Set with the **mfpdev app config language_preferences <value>** command.

For more details about the **mfpdev app config** command, type **mfpdev help app config** in your command window.

Description
Optional. Contains a comma-separated list of locales to display system messages.

Back to top

<mfp:clientCustomInit>

Syntax

```
<mfp:clientCustomInit enabled="false"/>
```

Contained in
`<widget>`

Configuration
Edited manually. You can set the `enabled` attribute value to either *true* or *false*.

Description
Controls how the `WL.Client.init` method is called. By default, this value is set to *false* and the `WL.Client.init` method is automatically called after the MobileFirst plug-in is initialized. Set this value to *true* for the client code to explicitly control when `WL.Client.init` is called.

Attributes

- `enabled`: Valid values are *true* and *false*.

Back to top

<mfp:server>

Syntax

```
<mfp:server
          url="http://10.0.0.1:9080"
          runtime="mfp" />
```

Contained in
`<widget>`

Configuration

- The server `url` value is set with the **mfpdev app config server** command.
- The server `runtime` value is set with the **mfpdev app config runtime** command.

For more details about the **mfpdev app config** command, type **mfpdev help app config** in your command window.

Description
Default remote server connection information, which the client application uses to communicate with the MobileFirst Server.

Attributes

- `url`: The `url` value specifies the MobileFirst Server protocol, host, and port values that the client will use to connect to the server by default.

- `runtime`: The `runtime` value specifies the MobileFirst Server runtime to which the application was registered. For more information about the MobileFirst runtime, see "MobileFirst Server overview" on page 6-2.

Back to top

`<mfp:ios>`

Syntax

```
<mfp:ios>
   <mfp:appChecksum>...</mfp:appChecksum>
   <mfp:sdkChecksum>...</mfp:sdkChecksum>
   <mfp:security>
      <mfp:testWebResourcesChecksum/>
   </mfp:security>
</mfp:ios>
```

Contained in
`<widget>`

Contains
- `<mfp:appChecksum>`
- `<mfp:sdkChecksum>`
- `<mfp:security>`

Description
This element contains all MobileFirst-related client application configuration for the iOS platform.

Back to top

`<mfp:android>`

Syntax

```
<mfp:android
   <mfp:appChecksum>...</mfp:appChecksum>
   <mfp:sdkChecksum>...</mfp:sdkChecksum>
   <mfp:security>
      <mfp:testWebResourcesChecksum/>
   </mfp:security>
</mfp:android>
```

Contained in
`<widget>`

Contains
- `<mfp:appChecksum>`
- `<mfp:sdkChecksum>`
- `<mfp:security>`

Description
This element contains all MobileFirst-related client application configuration for the Android platform.

Back to top

`<mfp:windows>`

Syntax

```
<mfp:windows>
   <mfp:appChecksum>...</mfp:appChecksum>
   <mfp:windowsphone8>
      <mfp:sdkChecksum>...</mfp:sdkChecksum>
      <mfp:security>
         <mfp:testWebResourcesChecksum/>
      </mfp:security>
   </mfp:windowsphone8>
   <mfp:windows8>
      <mfp:sdkChecksum>...</mfp:sdkChecksum>
      <mfp:security>
         <mfp:testWebResourcesChecksum/>
      </mfp:security>
        </mfp:windows8>
   <mfp:windows10>
      <mfp:sdkChecksum>...</mfp:sdkChecksum>
      <mfp:security>
         <mfp:testWebResourcesChecksum/>
      </mfp:security>
        </mfp:windows10>
</mfp:windows>
```

Contained in

`<widget>`

Contains

- `<mfp:appChecksum>`
- `<mfp:windowsphone8>`
- `<mfp:windows8>`
- `<mfp:windows10/>`

Description

This element contains all MobileFirst-related client application configuration for the Windows platforms.

Back to top

`<mfp:windows8>`

Syntax

```
<mfp:windows8>
   <mfp:sdkChecksum>...</mfp:sdkChecksum>
   <mfp:security>
      <mfp:testWebResourcesChecksum/>
   </mfp:security>
</mfp:windows8>
```

Contained in

`<mfp:windows>`

Contains

- `<mfp:sdkChecksum>`
- `<mfp:security>`

Description

This element contains all MobileFirst-related client application configuration for Windows 8.1 platforms.

Note: Windows Phone 8.1 configuration is contained under the `<mfp:windowsphone8>` element.

Back to top

<mfp:windowsphone>

Syntax

```
<mfp:windowsphone8>
   <mfp:sdkChecksum>...</mfp:sdkChecksum>
   <mfp:security>
      <mfp:testWebResourcesChecksum/>
   </mfp:security>
</mfp:windowsphone8>
```

Contained in
`<mfp:windows>`

Contains

- `<mfp:sdkChecksum>`
- `<mfp:security>`

Description
This element contains all MobileFirst-related client application configuration for the Windows Phone 8.1 platform.

Back to top

<mfp:windows10>

Syntax

```
<mfp:windows10>
   <mfp:sdkChecksum>...</mfp:sdkChecksum>
   <mfp:security>
      <mfp:testWebResourcesChecksum/>
   </mfp:security>
</mfp:windows10>
```

Contained in
`<mfp:windows>`

Contains

- `<mfp:sdkChecksum>`
- `<mfp:security>`

Description

This element contains all MobileFirst-related client application configuration for Windows 10 Universal Windows Platform (UWP).

Back to top

<mfp:appChecksum>

Syntax

```
<mfp:appChecksum>1234567890</mfp:appChecksum>
```

Contained in
`<mfp:ios>`, `<mfp:android>`, and `<mfp:windows>`.

Configuration
Not user-configurable. The checksum value is updated when the **mfpdev app webupdate** command is run.

For more details about the **mfpdev app webupdate** command, type **mfpdev help app webupdate** in your command window.

Description

This value is the checksum of application web resources. It is calculated when **mfpdev app webupdate** is run.

Back to top

<mfp:sdkChecksum>

Syntax

```
<mfp:sdkChecksum>2101152546</mfp:sdkChecksum>
```

Contained in

<mfp:ios>, <mfp:android>, <mfp:windowsphone8>, <mfp:windows8>, and <mfp:windows10>.

Configuration

Not user-configurable. This value is set by default.

Description

This value is the IBM MobileFirst Platform SDK checksum that is used to identify unique IBM MobileFirst Platform SDK levels.

Back to top

<mfp:security>

Syntax

```
<mfp:security>
   <mfp:testWebResourcesChecksum
      enabled="false"
      ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3/>
</mfp:security>
```

Contained in

<mfp:ios>, <mfp:android>, <mfp:windowsphone8>, <mfp:windows8>, and <mfp:windows10>.

Contains

<mfp:testWebResourcesChecksum/>

Description

This element contains the client application's platform-specific configuration for MobileFirst security.

Back to top

<mfp:testWebResourcesChecksum>

Syntax

```
<mfp:testWebResourcesChecksum
   enabled="false"
   ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
```

Contained in

<mfp:security>

Configuration

- The enabled attribute is set with the **mfpdev app config android_security_test_web_resources_checksum <value>** command.
- The ignoreFileExtensions attribute is set with the **mfpdev app config android_security_ignore_file_extensions <value>** command.

For more details about the **mfpdev app config** command, type **mfpdev help app config** in your command window.

Description
Controls whether the application verifies the integrity of its web resources each time it starts running on the mobile device.

Attributes

- `enabled`: Valid values are *true* and *false*. If this attribute is set to true, the application calculates the checksum of its web resources and compares this checksum with a value that was stored when the application was first run.
- `ignoreFileExtensions`: Checksum calculation can take a few seconds, depending on the size of the web resources. To make it faster, you can provide a list of file extensions to be ignored in this calculation. This value is ignored when the `enabled` attribute value is *false*.

Back to top

Cordova app resources:

You need certain resources as part of your Cordova app. In most cases, they are generated for you when you create your Cordova app with your preferred Cordova development tools. If you use the IBM MobileFirst Platform Foundation template, then splash screens and icons are also provided.

You can use a project template that is provided by IBM for use with Cordova projects that are enabled to use MobileFirst features. If you use this MobileFirst template, the following resources are made available to you as a starting point. If you do not use the MobileFirst template, all of the resources are provided except splash screens and icons.

You add the template by specifying the **--template** option and the MobileFirst template when you initially create your Cordova project. For more information about using this template, see "Creating a new Cordova app with the MobileFirst template" on page 7-88.

If you change the default file names and paths of any resources, you must also specify such changes in the Cordova configuration file (`config.xml`). In addition, in some cases, you can change the default names and paths with the **mfpdev app config** command. If you can change names and paths with the **mfpdev app config** command, it is noted in the section about the specific resource.

Cordova configuration file (config.xml)

The Cordova configuration file is a required XML file that contains application metadata and is stored in the root directory of the app. The file is automatically generated when you create a Cordova application. You can modify it to add custom properties by using the **mfpdev app config** command. For more information about this file, see "The Cordova configuration file" on page 7-94.

Main file (index.html)

This main file is an HTML5 file that contains the application skeleton. This file loads all the web resources (scripts and stylesheets) that are necessary to define the general components of the application and to hook to required document events. You can find this file in the `<*your project name*>/www` directory.

You can change the name of this file with the **`mfpdev app config`** command.

Thumbnail image

The thumbnail image provides a graphical identification for the application on the MobileFirst Operations Console. It must be a square image, preferably of size 90 by 90 pixels.

A default thumbnail image is provided when you use the template. You can override the default image by using the same file name with a replacement image. You can find `thumbnail.png` in `<`*`your project name`*`>/www/img`.

You can change the name or path of this file with the **`mfpdev app config`** command.

Splash image

The splash image is displayed while the application is being initialized.

If you use the MobileFirst default template, splash images are provided. These default images are stored in the following directories:

- iOS: `<`*`your project name`*`>/res/screen/ios/`
- Android: `<`*`your project name`*`>/res/screen/android/`
- Windows: `<`*`your project name`*`>/res/screen/windows/`

Various default splash images are included that are appropriate for different displays, and for iOS and Windows, different versions of the operating system. You can replace the default image that is provided by the template with your own splash image, or add an image if you did not use the template.

When you use the MobileFirst template to build your app for the Android platform, the `cordova-plugin-splashscreen` plug-in is installed. When this plug-in is integrated, the splash images that Cordova uses are displayed instead of the images that are used by MobileFirst. The images in the folder with the `screen.png` format are the Cordova standard splash images. You can specify which splash images display by changing the settings in the Cordova `config.xml` file.

If you do not use the MobileFirst template, the default splash images that are displayed are the images that are used by the MobileFirst plug-in. The file names of the default MobileFirst source splash images are in the form `splash-`*`string`*`.9.png`.

For more information about using your own splash images, see "Adding custom splash screens and icons to Cordova apps" on page 7-104.

Application icons

Default images for application icons are provided with the template. These default images are stored in the following directories:

- iOS: `<`*`your project name`*`>/res/icon/ios/`
- Android: `<`*`your project name`*`>/res/icon/android/`
- Windows: `<`*`your project name`*`>/res/icon/windows/`

You can replace the default image with your own image. Your custom application image must match the size of the default application image that you are replacing, and must use the same file name. Various default images are provided, appropriate to different displays and operating system versions. For more information about using your own icon images, see "Adding custom splash screens and icons to Cordova apps."

Stylesheets

The app code can include CSS files to define the application view.

The stylesheet files are located in the `<your project name>/www/css` directory, and are copied to the following platform-specific folders:
- iOS: `<your project name>/platforms/ios/www/css`
- Android: `<your project name>/platforms/android/assets/www/css`
- Windows: `<your project name>/platforms/windows/www/css`

Scripts

Your app code can include JavaScript files that implement various functions of your app such as interactive user interface components, business logic, and back-end query integration.

The JavaScript file `index.js` is provided by the template, and is located in the `<your project name>/www/js` folder. This file is copied to the following platform-specific folders:
- iOS: `<your project name>/platforms/ios/www/js`
- Android: `<your project name>/platforms/android/assets/www/js`
- Windows: `<your project name>/platforms/windows/assets/www/js`

Adding custom splash screens and icons to Cordova apps:

You can supply your own splash screens or icons in Cordova apps. You can add them, or replace existing images that are provided with the MobileFirst template, the `cordova-plugin-mfp` plug-in, or the `cordova-plugin-splashscreen`.

About this task

If you used the `cordova-plugin-mfp` plug-in, and you did not use the MobileFirst template or add the Cordova `cordova-plugin-splashscreen` plug-in to your app, you can replace the images for icons and splash screens that are provided by IBM MobileFirst Platform Foundation with your own images. If you used the template, then you can replace the splash images that the Cordova app uses, as they are the files that are displayed.

You must create a new folder to hold the splash images and icons, and modify the `config.xml` configuration file to point to them.

Procedure
1. Create a folder inside the root directory of your Cordova project. The folder can be in any level of nested subfolder when the parent folder is under the Cordova project root.
2. Place your source splash image and icon images in this folder.
3. Update the `config.xml` configuration file to point to your custom files.

- If you have an Android app, the requirements to for identifying the splash images depends on whether you created the app with or without the MobileFirst template.

  Splash screens
  If you did *not* use the MobileFirst template when you created your app, the splash images that are displayed are those that are retrieved from the MobileFirst images location. The target file paths and file names must remain exactly as in the example when you do not use the template. Change the source paths and file names (`src`) to the path of the files that you want to display. Add lines similar to the following example between the `<platform name="android">` and `</platform>` tags in the `config.xml` file:

```
<update src="res/screen/android/splash-hdpi.9.png" target="res/drawable-hdpi/splash.9.png" />
<update src="res/screen/android/splash-ldpi.9.png" target="res/drawable-ldpi/splash.9.png" />
<update src="res/screen/android/splash-mdpi.9.png" target="res/drawable-mdpi/splash.9.png" />
<update src="res/screen/android/splash-xhdpi.9.png" target="res/drawable-xhdpi/splash.9.png" />
<update src="res/screen/android/splash-xxhdpi.9.png" target="res/drawable-xxhdpi/splash.9.png" />
```

  If you used the MobileFirst template when you created your app, you must update the splash images that Cordova uses. Change the source paths and file names (`src`) to the path of the files that you want to display. Add lines similar to the following example between the `<platform name="android">` and `</platform>` tags in the `config.xml` file:

```
<splash density="land-hdpi" src="res/screen/android/screen-hdpi-landscape.png" />
<splash density="land-ldpi" src="res/screen/android/screen-ldpi-landscape.png" />
<splash density="land-mdpi" src="res/screen/android/screen-mdpi-landscape.png" />
<splash density="land-xhdpi" src="res/screen/android/screen-xhdpi-landscape.png" />
<splash density="hdpi" src="res/screen/android/screen-hdpi-portrait.png" />
<splash density="ldpi" src="res/screen/android/screen-ldpi-portrait.png" />
<splash density="mdpi" src="res/screen/android/screen-mdpi-portrait.png" />
<splash density="xhdpi" src="res/screen/android/screen-xhdpi-portrait.png" />
```

  Icons The file names of the icon files must be the same as the entries in the following example. The paths can be any path. The name of each image corresponds to its size.

```
<icon src="res/icon/android/icon-96-xhdpi.png" />
<icon density="mdpi" src="res/icon/android/icon-48-mdpi.png" />
<icon density="hdpi" src="res/icon/android/icon-72-hdpi.png" />
<icon density="xhdpi" src="res/icon/android/icon-96-xhdpi.png" />
<icon density="xxhdpi" src="res/icon/android/icon-144-xxhdpi.png" />
<icon density="xxxhdpi" src="res/icon/android/icon-192-xxxhdpi.png" />
```

- If you have an iOS app, add lines similar to the following example between the `<platform name="ios">` and `</platform>` tags:

  Splash screens

  The paths and file names of the splash screen files must be the same as the names in the following example. The name of each image corresponds to its size.

```
<splash height="480" src="res/screen/ios/Default⌂iphone.png" width="320" />
<splash height="1024" src="res/screen/ios/Default-Portrait⌂ipad.png" width="768" />
<splash height="2048" src="res/screen/ios/Default-Portrait@2x⌂ipad.png" width="1536" />
<splash height="768" src="res/screen/ios/Default-Landscape⌂ipad.png" width="1024" />
<splash height="1536" src="res/screen/ios/Default-Landscape@2x⌂ipad.png" width="2048" />
<splash height="1136" src="res/screen/ios/Default-568h@2x⌂iphone.png" width="640" />
<splash height="1334" src="res/screen/ios/Default-667h⌂iphone.png" width="750" />
<splash height="2208" src="res/screen/ios/Default-736h⌂iphone.png" width="1242" />
<splash height="1242" src="res/screen/ios/Default-736h-Landscape⌂iphone.png" width="2208" />
```

  Icons

The file names of the icon files must be the same as the names in the following example. The paths can be any path. The name of each image corresponds to its size.

```
<icon height="167" src="res/icon/ios/icon-83.5@2x.png" width="167"/>
<icon height="57" src="res/icon/ios/icon.png" width="57" />
<icon height="114" src="res/icon/ios/icon@2x.png" width="114" />
<icon height="40" src="res/icon/ios/icon-40.png" width="40" />
<icon height="80" src="res/icon/ios/icon-40@2x.png" width="80" />
<icon height="50" src="res/icon/ios/icon-50.png" width="50" />
<icon height="100" src="res/icon/ios/icon-50@2x.png" width="100" />
<icon height="60" src="res/icon/ios/icon-60.png" width="60" />
<icon height="120" src="res/icon/ios/icon-60@2x.png" width="120" />
<icon height="180" src="res/icon/ios/icon-60@3x.png" width="180" />
<icon height="72" src="res/icon/ios/icon-72.png" width="72" />
<icon height="144" src="res/icon/ios/icon-72@2x.png" width="144" />
<icon height="76" src="res/icon/ios/icon-76.png" width="76" />
<icon height="152" src="res/icon/ios/icon-76@2x.png" width="152" />
<icon height="29" src="res/icon/ios/icon-small.png" width="29" />
<icon height="58" src="res/icon/ios/icon-small@2x.png" width="58" />
<icon height="87" src="res/icon/ios/icon-small@3x.png" width="87" />
```

- If you have a Windows app, add lines similar to the lines in the following example between the `<platform name="windows">` and `</platform>` tags:

  Splash screens

  The paths and file names of the splash screen files must be the same as the names in the following example. The name of each image corresponds to its size.

```
<splash src="res/screen/windows/SplashScreen.scale-100.png" width="620" height="300"/>
<splash src="res/screen/windows/SplashScreenPhone.scale-240.png" width="1152" height="1920"/>
<splash src="res/screen/windows/Wide310x150Logo.scale-100.png" width="310" height="150"/>
<splash src="res/screen/windows/Wide310x150Logo.scale-240.png" width="744" height="360"/>
```

  Icons

  The file names of the icon files must be the same as the names in the following example. The paths can be any path. The name of each image corresponds to its size.

```
<icon src="res/icon/windows/Square30x30Logo.scale-100.png" width="30" height="30" />
<icon src="res/icon/windows/Square44x44Logo.scale-100.png" width="44" height="44" />
<icon src="res/icon/windows/Square44x44Logo.scale-240.png" width="106" height="106" />
<icon src="res/icon/windows/Square70x70Logo.scale-100.png" width="70" height="70" />
<icon src="res/icon/windows/Square71x71Logo.scale-100.png" width="71" height="71" />
<icon src="res/icon/windows/Square71x71Logo.scale-240.png" width="170" height="170" />
<icon src="res/icon/windows/Square150x150Logo.scale-100.png" width="150" height="150" />
<icon src="res/icon/windows/Square150x150Logo.scale-240.png" width="360" height="360" />
<icon src="res/icon/windows/Square310x310Logo.scale-100.png" width="310" height="310" />
<icon src="res/icon/windows/StoreLogo.scale-100.png" width="50" height="50" />
<icon src="res/icon/windows/StoreLogo.scale-240.png" width="120" height="120" />
```

What to do next

For more information about splash images and icons, see the Apache Cordova page about splash images.

Registering Cordova applications to MobileFirst Server

Register your Cordova application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app. You can register your app by using the IBM MobileFirst Platform Command Line Interface (CLI) or the IBM MobileFirst Platform Operations Console.

Once you have registered your client application on the server, your app can access the server resources. For details on how to test the server registration from your client app see "Some initial WebView code for connecting to the server" on page 7-130.

Note: If you do not register before running the app, the `mfpclient` file is not created in the `[platform]\assets` folder and the following error appears at runtime:

```
java.lang.RuntimeException: Client configuration file mfpclient.properties not found
in application assets. Use the MFC CLI command 'mfpdev app register' to create the file.
```

You must register your app before running it. If you want to run the app before registering (for example if you have no server), you can create the `mfpclient` file by running **cordova prepare** from the Cordova app root folder.

However this `mfpclient` file does not contain the necessary connection values. You cannot run an application that includes requests to server resources, without properly registering the app.

To view the initial app in an emulator, after creating it with Cordova and the MobileFirst plug-in, you can run it after running **cordova prepare**.

Registering Cordova applications from the MobileFirst Platform CLI:

You can use the IBM MobileFirst Platform Command Line Interface (CLI) to register your Cordova application to an instance MobileFirst Server.

Before you begin

- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have an instance of MobileFirst Server running. The server can be running locally, or it can be a remote server, but it must be reachable from your local computer. For more information, see "Setting up the MobileFirst Development Server" on page 7-12, "Installing the IBM MobileFirst Platform Foundation Developer Kit" on page 7-10, or "Installing IBM MobileFirst Platform Server" on page 6-2.
- You must have a Cordova application on your local computer, with platforms already installed. For more information, see The Command-Line Interface at the Apache Cordova web site.

About this task

Once you have the client side of your Cordova application initially defined, you can prepare for further development tasks by registering it to a MobileFirst Server.

Procedure

1. Check that the target MobileFirst Server is up and running.
2. Navigate to the directory that contains your app, or one of its subdirectories.

   Important:

   - If the current directory is `platforms/`*`platform`* or one of its subdirectories, the registration occurs for only the corresponding platform.

- For the Windows platform, if you want to register only specific versions of the Windows app, use the `--windows` option of the **`mfpdev app register`** command to specify the versions. For example: `mfpdev app register --windows windows,windowsphone8` registers the app for Windows 10 Universal and Windows Phone 8.1. `mfpdev app register --windows windows8` registers the Windows 8.1 desktop portion of your Cordova app.

3. Register your app to the server. Use one of the following procedures:
   - To register the app to the default server, run the following command:
     ```
     mfpdev app register
     ```
     Note: If you have not previously set a default server and a MobileFirst Server is running on the local system, this command registers the app to the local MobileFirst Server, and this server is made the default.
   - To register your app to a server that is not the default server:
     a. Create a server profile by running the **`mfpdev server add`** command. For example:
        ```
        mfpdev server add Server1 -url https://company.mobile.com:9080 -login admin -password secre
        ```
        For more information about the **`mfpdev server add command`**, run `mfpdev help server add`.
     b. To register your app to the server that you just defined, run the **`mfpdev app register`** command, and specify the server profile that you just created. For example:
        ```
        mfpdev app register Server1
        ```
   For more information about this command, including optional parameters, run **`mfpdev help app register`**.
4. To propagate the changes to all the target platforms of your Cordova app, run the **`cordova prepare`** command.. The server URL is copied into the `config.xml` files that are located in each platform's subdirectory.

Results

The app is registered to the target server. Data about the app that is obtained from the `config.xml` file such as application name, version number, and app ID is sent to the server. The root client properties file `config.xml` is updated with the value of the server's URL, as are the copies of `config.xml` that reside in subdirectories that correspond to each of the app's platforms.

What to do next

You can proceed with other development tasks that depend on the MobileFirst Server. For example, you can preview your app, test your app's security features, and manage your app from the MobileFirst Operations Console.

Registering Cordova apps from the MobileFirst Operations Console:

Register your Cordova application to an instance of MobileFirst Server to establish communication with the server and to provide the server with information about your app. You must register your app to the server before running or testing code that accesses server resources.

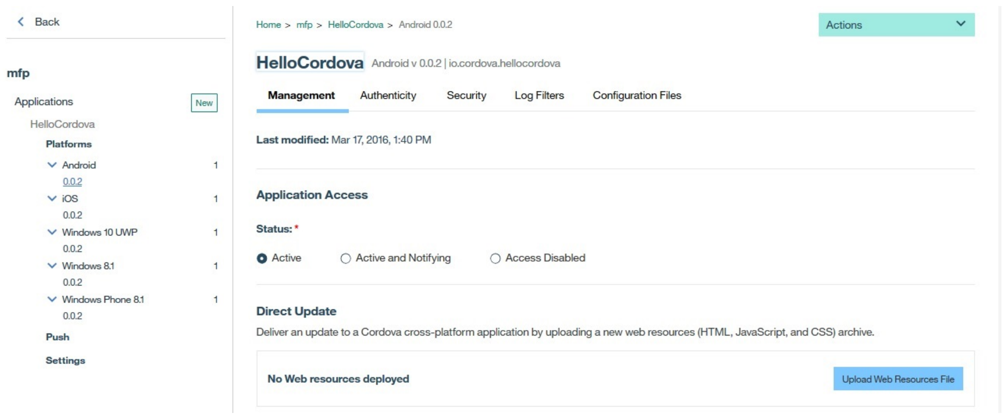
Before you begin

You must have the IBM MobileFirst Platform Operations Console running on the MobileFirst Server targeted for registration. For more information, see “The IBM MobileFirst Platform Foundation Developer Kit” on page 7-9.

About this task

Each platform of a Cordova app is registered as an independent application on the MobileFirst Server. However in the IBM MobileFirst Platform Operations Console they are grouped together under a single display name and application id. To register your Cordova MobileFirst app using IBM MobileFirst Platform Operations Console, register each platform according to the native app registration procedure, using the same display name and application id for each platform. For information on how to register each app see “Registering iOS applications from the MobileFirst Operations Console” on page 7-38, “Registering Android applications from the MobileFirst Operations Console” on page 7-60, or “Registering Windows applications from the MobileFirst Operations Console” on page 7-70.

Each platform appears as a separate app in the console, under a single display name and application id in the left pane, regardless of whether the Cordova app was registered with IBM MobileFirst Platform Operations Console or with MobileFirst Platform CLI.



< Back

mfp

Applications New

HelloCordova

Platforms

Android 1
0.0.2
iOS 1
0.0.2
Windows 10 UWP 1
0.0.2
Windows 8.1 1
0.0.2
Windows Phone 8.1 1
0.0.2

Push

Settings

Home > mfp > HelloCordova > Android 0.0.2

Actions

HelloCordova Android v 0.0.2 | io.cordova.hellocordova

Management | Authenticity | Security | Log Filters | Configuration Files

Last modified: Mar 17, 2016, 1:40 PM

Application Access

Status: *

● Active ○ Active and Notifying ○ Access Disabled

Direct Update

Deliver an update to a Cordova cross-platform application by uploading a new web resources (HTML, JavaScript, and CSS) archive.

No Web resources deployed

Upload Web Resources File

Results

Your Cordova applications are registered on the server. Regardless of the method used for registering, each platform app needs to be managed separately in regard to the options provided by the console (management, security, authenticity, etc.).

Previewing Cordova web resources with the Mobile Browser Simulator

You can preview the web resources of your app and test some of your JavaScript code with Mobile Browser Simulator.

Restriction: The Mobile Browser Simulator supports the following web browsers:
- Firefox version 38 and later
- Chrome 49 and later
- Safari 9 and later

Restriction: You preview your web resources with Mobile Browser Simulator, but not all MobileFirst JavaScript APIs are supported by the simulator. In particular, the OAuth protocol is not fully supported. However, you can test calls to adapters with WLResourceRequest. In this case,

- Security checks are not run on the server-side and security challenges are not sent to the client that runs in Mobile Browser Simulator.
- If you do not use MobileFirst Development Server, register a confidential client that has the adapter's scope in its list of allowed scopes. You can define a confidential client with the MobileFirst Operations Console, by using the **Runtime/Settings** menu. For more information about confidential clients, see “Confidential clients” on page 7-279.

  Note: MobileFirst Development Server includes a confidential client "test" that has an unlimited allowed scope ("*"). By default **`mfpdev app preview`** uses this confidential client.

To test your application with a device simulator, use instead **`cordova emulate`**.

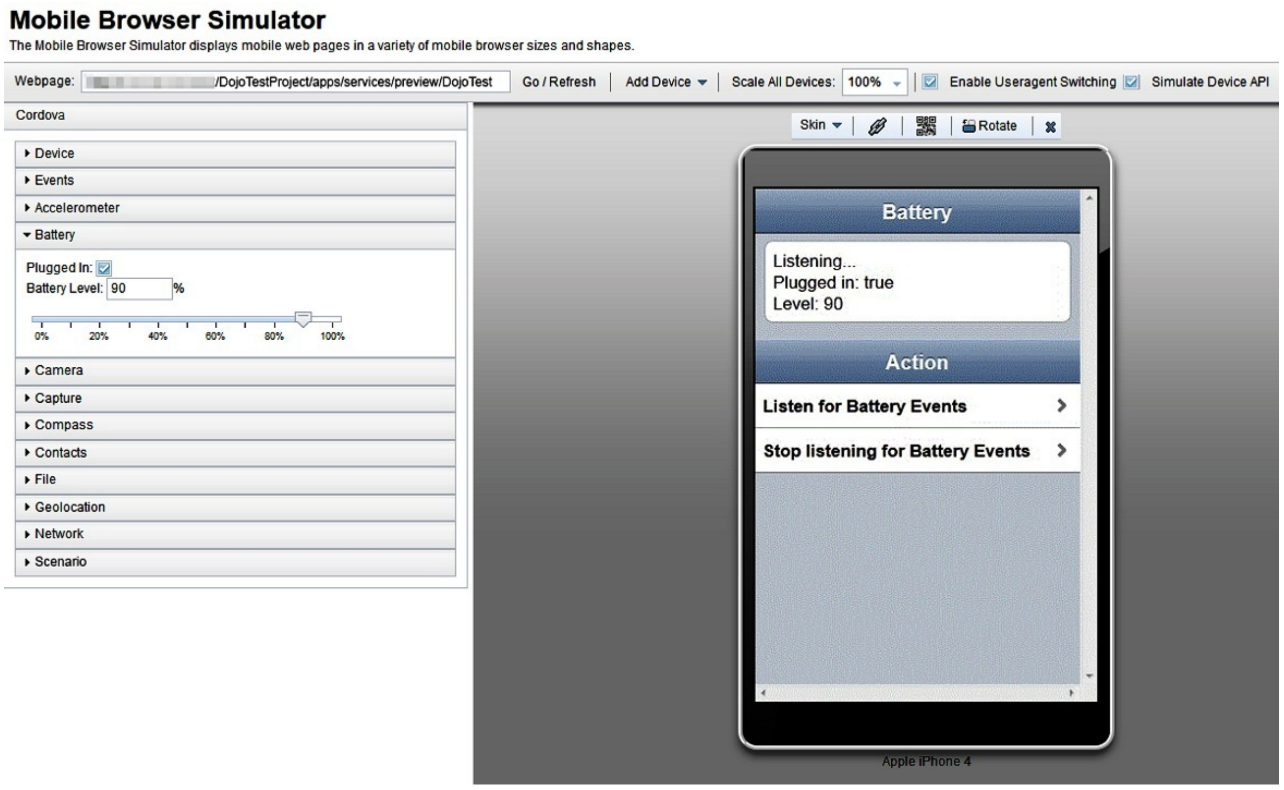
To preview your Cordova web resources with Mobile Browser Simulator, use the following procedure

1. If you plan to use the connection with the server, start your test server. You start MobileFirst Development Server with the **`run`** command in the server installation directory. For more information, see “Starting the MobileFirst Development Server” on page 7-12.
2. Use **`mfpdev app preview`**. For example, to preview the iOS platform of your application, with an update when you modify the web resource, type
   ```
   $ mfpdev app preview ios --type mbs -wp
   ```
   To preview the same application and test an adapter on a server that is not MobileFirst Development Server, type
   ```
   $ mfpdev app preview ios --type mbs -wp --clientid clientid --secret clientsecret
   ```
   Where
   - *clientid* is the ID of the confidential client to use
   - *cliensecret* is the secret of the confidential client to use

For more information about **`mfpdev app preview`**, you can type

```
$ mfpdev help app preview
```

The following figure shows the display of the Mobile Browser Simulator.



Cordova app security

IBM MobileFirst Platform Foundation provides security features that help you protect your Cordova apps.

Much of the content in a cross-platform app can be more easily modified by an unauthorized person than for a native app. Because much of the common content in a cross-platform app is in a readable format, IBM MobileFirst Platform Foundation provides features that can provide a higher level of security for your cross-platform Cordova apps. See "MobileFirst security framework" on page 7-265 for information about security options that are not mentioned in this topic. Use the following features to improve security on your Cordova apps:

"Encrypting the web resources of your Cordova packages" on page 7-112
: Encrypts the contents in the `www` folder of your Cordova app, and decrypts it when the app is installed and run for the first time. This encryption makes it more difficult for someone to view or modify the content in that folder while the app is packaged.

"Enabling the web resources checksum feature" on page 7-113
: Ensures the integrity of the app when it starts by comparing the contents to the baseline checksum results that were gathered the first time the app was started. This test helps prevent the modification of an app that is already installed.

"Enabling FIPS 140-2" on page 10-77
: Ensures that the encryption algorithms that are used to encrypt data at rest and data in motion are compliant with the Federal Information Processing Standards (FIPS) 140-2 standard.

"Certificate pinning" on page 7-185
: Helps you prevent man-in-the-middle attacks by associating a host with its expected public key.

Encrypting the web resources of your Cordova packages:

You can use the MobileFirst Platform CLI to encrypt the content in the `www` folder of the `.apk` or `.ipa` packages of your Cordova web apps.

Before you begin

- You must have the Cordova development tools installed. This example uses the Apache Cordova CLI. If you use other Cordova development tools, some of your steps will be different. Refer to your Cordova tool documentation for instructions.
- You must have the MobileFirst Platform CLI installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have the IBM MobileFirst Platform Foundation plug-in. See "Cordova plug-ins for MobileFirst features" on page 7-87 for more information about the plug-in.

The best time to complete this procedure is after finishing your app development and are ready to deploy the app. If you run any of the following commands after you complete the web resources encryption procedure, the content that was encrypted becomes decrypted:

- **`cordova prepare`**
- **`cordova build`**
- **`cordova run`**
- **`cordova emulate`**
- **`mfpdev app webupdate`**
- **`mfpdev app preview`**

If you run one of the listed commands after you encrypt the web resources, you must complete this procedure again to encrypt the web resources.

About this task

To minimize the risk of someone viewing and modifying your web resources while it is in the `.apk` or `.ipa` package, you can use the IBM MobileFirst Platform Foundation **`mfpdev app webencrypt`** command or the **`mfpwebencrypt`** flag to encrypt the information. This procedure does not provide encryption that is impossible to defeat, but it provides a basic level of obfuscation.

To encrypt the content in the package of your Cordova app with the MobileFirst Platform CLI, complete the following steps:

Procedure

1. Open a terminal window and navigate to the root directory of the Cordova app that you want to encrypt.
2. Prepare the app by entering one of the following commands:
   - `cordova prepare`
   - `mfpdev app webupdate`
3. Complete one of the following procedures to encrypt the content:
   - Enter the following command:

```
mfpdev app webencrypt
```

Tip: You can view information about the **mfpdev app webencrypt** command by entering **mfpdev help app webencrypt**.

- You can also encrypt the web resources of your Cordova packages by adding the mfpwebencrypt flag to the **cordova compile** or to the **cordova build** command when you build your packages.

  ```
  cordova compile -- --mfpwebencrypt
  ```

  ```
  cordova build -- --mfpwebencrypt
  ```

The operating system information in the www folder is replaced by a resources.zip file that contains the encrypted content.

If your app is for the Android operating system and the resources.zip file is larger than 1 MB, the resources.zip file is divided into smaller 768 KB .zip files that are named resources.zip.*nnn*. The variable *nnn* is a number from 001 through 999.

4. Test the application with the encrypted resources by using the emulator that is provided with the platform-specific tools. For example, you can use the emulator in Android Studio for Android, or Xcode for iOS.

   Note: Do not use the following Cordova commands to test the application after you encrypt it:
   - **cordova run**
   - **cordova emulate**

   These commands refresh the content that was encrypted in the www folder, and saves it again as decrypted content. If you use these commands, remember to complete the procedure again to encrypt it before you publish the app.

Enabling the web resources checksum feature:

Use the web resources checksum feature to verify the integrity of your cross-platform app.

Before you begin

- You must have the Cordova development tools installed. This example uses the Apache Cordova CLI. If you use other Cordova development tools, some of your steps will be different. Refer to your Cordova tool documentation for instructions.
- You must have the MobileFirst Platform CLI installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.
- You must have the IBM MobileFirst Platform Foundation plugin.
- You must add the platform to your Cordova project before you can enable the web resources checksum feature for that operating system by entering the `cordova platform add [`*android*`|`*ios*`|`*windows*`]` command.

About this task

When it is enabled, the web resources checksum feature compares the original web resources of an app when it is started to a stored baseline that was captured the first time that app was started. This is a good way of identifying any differences in the app that might indicate that the app was modified. This procedure is compatible with the Direct Update feature.

To enable the web resources checksum feature for a Cordova app, complete the following steps:

Procedure

1. In a terminal window, navigate to the root directory of your target app.
2. Enter the following command to enable the web resources checksum feature for an operating system environment of your Cordova app:
   ```
   mfpdev app config [android|ios|windows10|windows8|
   windowsphone8]_security_test_web_resources_checksum true
   ```
   For example:
   ```
   mfpdev app config android_security_test_web_resources_checksum true
   ```
   You can disable the feature by replacing *true* in the command with *false*.

   Tip: You can view information about the **mfpdev app config** command by entering **mfpdev help app config**.
3. Enter the following command to identify the types of files that you want to ignore during the checksum test:
   ```
   mfpdev app config [android|ios|windows10|windows8|
   windowsphone8]_security_ignore_file_extensions [ file_extension1,file_extension2 ]
   ```
   Multiple extensions must be separated by a comma with no spaces between them. For example:
   ```
   mfpdev app config android_security_ignore_file_extensions jpg,png,pdf
   ```
   Important: Running this command overwrites the values that are set.
   The more files that the web resources checksum scans for its test, the longer it takes for the app to open. You can specify the extension of a file type to skip, which might improve the speed of starting the app.

Results

Your app has the web resources checksum feature enabled.

What to do next

1. Run the following command to integrate the changes into your app:
   ```
   cordova prepare
   ```
2. Build your app by entering the following command:
   ```
   cordova build
   ```
3. Run your app by entering the following command:
   ```
   cordova run
   ```

Related concepts:

"Command-line interface (CLI) help" on page 7-18
You can run the **help** command to display full information about all MobileFirst Platform CLI commands.

IBM MobileFirst Studio plug-in for managing Cordova projects in Eclipse

The IBM MobileFirst Studio plug-in helps you manage your Cordova project in the Eclipse development environment.

Cordova projects are typically managed by entering commands on the Cordova command line. After integrating the IBM MobileFirst Studio plug-in into your Eclipse environment, you can run the following commands on your Cordova app without leaving the Eclipse development environment:

- Open server console
- Preview app
- Register app
- Encrypt app
- Pull app
- Push app
- Update app

You can also open your project in Android Studio or Xcode directly from the Eclipse interface.

Integrating the MobileFirst Studio plug-in to manage a Cordova project in Eclipse:

You can integrate the IBM MobileFirst Studio plug-in into your Eclipse development environment to run extra commands on your Cordova projects from an Eclipse menu.

Before you begin

Be sure that you have the following prerequisites before you start the procedure:

- A Mars Java EE version of Eclipse, or later.
- Java JRE version 7, or later.
- Node.js version 4.x, or later.
- The Cordova command-line interface, version 6.1.1, or later.
- The IBM MobileFirst Platform Command Line Interface (CLI). For more information, see “The MobileFirst command-line interface (CLI)” on page 7-13.
- An internet connection.

About this task

This procedure includes instructions for setting up the IBM MobileFirst Studio plug-in for Eclipse with or without The Eclipse Hybrid Mobile (THyM) version 2.0 plug-in integrated with your Eclipse environment. The THyM plug-in simplifies some of the management tasks of your Cordova projects in the Eclipse environment. These tasks include installing Cordova plug-ins directly from the Eclipse interface and creating Cordova projects directly in the Eclipse environment. The instructions apply to both situations, so follow the procedure that applies to your situation.

To integrate the IBM MobileFirst Studio plug-in into your Eclipse environment and manage a Cordova project in Eclipse, complete the following steps:

Procedure

1. Install the IBM MobileFirst Studio Eclipse plug-in.
   a. Search for the `IBM MobileFirst Studio Plug-in` in the Eclipse marketplace.
   b. Select `IBM MobileFirst Studio Plugins`, and click **Next**.
   c. Accept the license terms and click **Finish**.

   d. Restart Eclipse.
2. Optional: With the THyM plug-in: Install the THyM plug-in. The THyM plug-in is an optional open source plug-in that helps you manage your Cordova app in the Eclipse development environment. THyM is not required to use the IBM MobileFirst Studio plug-in, but it simplifies some of the Cordova project management within the Eclipse environment. IBM does not support issues that result from the THyM plug-in. Steps that refer to the THyM plug-in are provided as a convenience.
   a. Open your Eclipse development environment and select **Help** > **Eclipse Marketplace**.
   b. Search for the plug-in for THyM 2.0, or later, and click **Install**.

      Restriction: Only THyM version 2.0, or later, is supported.
   c. Confirm that you want to install the THyM plug-in by ensuring that the check boxes for **Eclipse Thym 2.0.0** and **Hybrid Mobile Application Development Tools** are selected. Click **Confirm**.
   d. Read and accept the license agreement; then, click **Finish**.
   e. Restart Eclipse.
3. Create your Cordova project, if necessary. If you already have an existing project that you are importing, skip to step 4 on page 7-117 if you are not using the THyM plug-in procedure, or to steps 5 on page 7-117 or 6 on page 7-117 if you are following the procedure with the THyM plug-in.
   - With the THyM plug-in:
     a. Select **File** > **New** > **Other** > **Mobile** > **Hybrid Mobile (Cordova) Application Project**, and click **Next**.
     b. Specify a name for your project, and click **Next**.
     c. Specify each platform, or engine, that your app uses, and click **Finish**. The platforms that are supported in IBM MobileFirst Platform Foundation are Android, iOS, and Windows Universal.

        Tip: Click **Download...** to retrieve any engines that are not available to select.
   - Without the THyM plug-in:
     a. In your terminal window, go to the directory where you want to create your Cordova app.
     b. Create your Cordova app by entering the following command:
        ```
        cordova create app_name app_identifier directory_for_app
        ```

        where the following are true:

        app_name
            The name of your app.

        app_identifier
            The unique identifier for your app.

        directory_for_app
            The directory where you want to create your app.
     c. In your terminal window, change to the root directory of your Cordova app.
     d. Add the platforms for your app by entering the following command:
        ```
        cordova platform add [ios | android | windows]
        ```

Repeat this step for each platform that you want to add.

4. Without the THyM plug-in only: Add the IBM MobileFirst Platform Foundation Cordova plug-in to your Cordova project.
   a. Go to the root directory of your Cordova project in the terminal window.
   b. Enter the following command to add the plug-in:
   ```
   cordova plugin add cordova-plugin-mfp
   ```
5. Import your project into Eclipse, if necessary. If your existing Cordova project was created in Eclipse with the THyM plug-in installed, then you do not have to import it again.
   - With the THyM plug-in:
     a. Open the Eclipse development environment, and create a new project by selecting: **File** > **Import** > **General** > **Import Cordova project**, and click **Next**.
     b. Browse to the location of the project. Select it and click **OK**.
     c. Select the project and click **Finish**.
   - Without the THyM plug-in:
     a. Open the Eclipse development environment, and create a new project by selecting: **File** > **New** > **Project** > **General** > **Project**, and click **Next**.
     b. Enter a name for your project.
     c. Clear the check box for `Default location`.
     d. Browse to the root directory of your existing Cordova project.
     e. Select the Cordova project and click **Finish**.

        Note: If your Cordova project is in the same path as your Eclipse workspace, the **Finish** button is not active.

     Important: The IBM MobileFirst Platform Foundation Cordova plug-in must be installed in your Cordova project when you import it into Eclipse when you are not using the THyM plug-in. See step 4 for instructions about installing the IBM MobileFirst Cordova plug-in when you are not using the THyM plug-in.
6. With the THyM plug-in only: Install the IBM MobileFirst Platform Foundation Cordova plug-in to your project in Eclipse.
   a. Right-click the name of your Cordova project in the Eclipse navigation.
   b. Select **Install a Cordova Plug-in**.
   c. Use the search string of "mfp" to find and install the `cordova-plugin-mfp`.
   d. Read and accept the license agreement; then, click **Finish**.
   e. Restart Eclipse.

Results

You can access the IBM MobileFirst Studio plug-in menu by right-clicking the name of your Cordova project in the Eclipse development environment. See Table 7-14 on page 7-118 for the commands that are supported and their actions.

Running MobileFirst Studio plug-in commands in Eclipse:

After you integrate the IBM MobileFirst Studio plug-in into your Eclipse development environment, you can run extra IBM MobileFirst Platform Foundation commands on your Cordova projects from within your Eclipse development environment.

Before you begin

Be sure that you have the following prerequisites before you start the procedure:
- A Mars Java EE version of Eclipse, or later, with the IBM MobileFirst Studio plug-in installed.
- Java JRE version 7, or later.
- Node.js version 4.x, or later.
- The Cordova command-line interface, version 6.1.1, or later.
- The IBM MobileFirst Platform Command Line Interface (CLI), version 8.0, or later. For more information, see "The MobileFirst command-line interface (CLI)" on page 7-13.
- An internet connection.
- A Cordova project that has the `cordova-plugin-mfp` installed.

Important: To use this function, you must have installed the IBM MobileFirst Studio plug-in as explained in "Integrating the MobileFirst Studio plug-in to manage a Cordova project in Eclipse" on page 7-115.

About this task

The IBM MobileFirst Studio plug-in commands that you run in Eclipse automatically run on the default server that is specified in your IBM MobileFirst Platform Command Line Interface (CLI) server settings. The default target server is the local server at: `http://localhost:9080/mfpadmin`. You can change the default server setting by using your IBM MobileFirst Platform Command Line Interface (CLI). For more information about how to define servers, see "Defining the target server of the MobileFirst Platform CLI" on page 7-16.

To run an IBM MobileFirst Studio plug-in command from the Eclipse menu, complete the following steps:

Procedure

1. Right-click the name of your Cordova project in your Eclipse navigation pane.
2. Select the IBM MobileFirst Studio plug-ins menu.
3. Select the command that you want to run from the following options:

Table 7-14. Commands that are available with the IBM MobileFirst Studio plug-in

IBM MobileFirst Studio plug-in menu option	**Action**	**IBM MobileFirst Platform Command Line Interface (CLI) equivalent** **Tip:** For more information about each of these commands, enter the specified command on the command line and add `help` after `mfpdev`.
Open server console	When the server definition exists, opens the console so you can view the actions of the specified server.	**`mfpdev server console`**
Preview app	Opens the app in the mobile browser (MBS) preview mode.	**`mfpdev app preview`**

Table 7-14. Commands that are available with the IBM MobileFirst Studio plug-in (continued)

IBM MobileFirst Studio plug-in menu option	Action	IBM MobileFirst Platform Command Line Interface (CLI) equivalent **Tip:** For more information about each of these commands, enter the specified command on the command line and add `help` after `mfpdev`.
Register app	Registers the app with the server that is specified in your server definitions.	**`mfpdev app register`**
Encrypt app	Runs the web resource encryption tool on your app.	**`mfpdev app webencrypt`**
Pull app	Retrieves the existing app configuration from the server that is specified in the server definition.	**`mfpdev app pull`**
Push app	Sends the app configuration of your current app to the server that is specified in the build definition so you can reuse it for another app.	**`mfpdev app push`**
Update app	Packages the contents of the `www` folder in a .zip file, and replaces the version on the server with the package.	**`mfpdev app webupdate`**

Setting the debug level for the MobileFirst Studio plug-in in Eclipse:

With the IBM MobileFirst Studio plug-in, you can view the log output information for your Cordova project in the Eclipse console and control the amount of information that is displayed.

Before you begin

Be sure that you have the following prerequisites before you start the procedure:
- A Mars Java EE version of Eclipse, or later, with the IBM MobileFirst Studio plug-in installed.
- Java JRE version 7, or later.
- Node.js version 4.x, or later.
- The Cordova command-line interface, version 6.1.1, or later.
- The IBM MobileFirst Platform Command Line Interface (CLI).
- An internet connection.
- A Cordova project that has the `cordova-plugin-mfp` plug-in installed.

Important: To use this function, you must have installed the IBM MobileFirst Studio plug-in as explained in "Integrating the MobileFirst Studio plug-in to manage a Cordova project in Eclipse" on page 7-115.

About this task

You can select to display either the normal log messages in the Eclipse console, or more verbose messaging that can help you isolate a problem. Because of the amount of detail that is provided when verbose debug mode is on, activate this mode only when you are troubleshooting a problem. To change the setting, complete the following steps:

Procedure

1. In your Eclipse development interface, select **Window** > **Preferences**. This mode displays verbose debug logs so you can view them in the Eclipse console window while you are previewing the app.
2. Select **MobileFirst Studio Plugins**.
3. Ensure that **Enable debug mode** is selected.
4. Apply the changes, and click **OK**. The verbose log messages are displayed in the Eclipse console.

Results

If selected, the debug mode is set to display verbose messages in the Eclipse console. If not selected, standard logging messages are displayed.

Opening a Cordova project in a platform development environment:

With the IBM MobileFirst Studio plug-in, you can open your Cordova project in the platform IDE directly from Eclipse.

Before you begin

Be sure that you have the following prerequisites before you start the procedure:
- A Mars Java EE version of Eclipse, or later, with the IBM MobileFirst Studio plug-in installed.
- Java JRE version 7, or later.
- Node.js version 4.x, or later.
- The Cordova command-line interface, version 6.1.1, or later.
- The IBM MobileFirst Platform Command Line Interface (CLI).
- An internet connection.
- A Cordova project with an iOS or Android platform, and the `cordova-plugin-mfp` plug-in installed.

Important: To use this function, you must install the IBM MobileFirst Studio plug-in as explained in "Integrating the MobileFirst Studio plug-in to manage a Cordova project in Eclipse" on page 7-115.

About this task

After you create a Cordova project and add the iOS or Android platform, you might want to open the project within the Xcode or Android Studio environment. You can open it directly from within the Eclipse environment if you have the IBM MobileFirst Studio plug-in.

Important: Do not edit the code in your projects in the Android Studio or Xcode environment. Changes that are made in the IDE only apply to the platform that is

open and can be overwritten by information at the project level.
To open the project in the native IDE, complete the following steps:

Procedure

1. Ensure that your path to Android Studio is set correctly by selecting **Preferences** > **MobileFirst Studio Plugins**.
   - In a Windows environment, specify the directory that contains the `bin` file, which is often `C:\Program Files\Android\Android Studio`.
   - In an OSX environment, specify the directory that contains the Android Studio application, which is often `/Applications`.
2. Import your Android project into Android Studio before launching it from Eclipse. You only have to do this the first time that you are launching the Android project from Eclipse.
   a. Open Android Studio.
   b. Select **File** > **New** > **Import project...**.
   c. Select the project folder, the `build.gradle` file, or the `settings.gradle` file for the project that you want to import. You can usually find the `build.gradle` file and the `settings.gradle` file in the `c:\dir\`*`project_name`*`\platforms\android` directory.

   You can open your Cordova project that contains the Android platform in Android Studio directly from the Eclipse IDE.
3. Open your Cordova project in the Eclipse development environment.
4. Right-click the project and select **Run as...** > **Xcode project** or **Run as...** > **Android Studio project** to open the project in the native environment. If you are not in an OSX environment, then the **Xcode project** option is not available.

Results

The project opens in the selected development environment.

Developing Cordova apps for Android

The Cordova app can run on the Android platform. Once the app is initiated using the native code of the Android OS, the WebView is loaded and the MobileFirst integration is accessed through the JavaScript API.

The MobileFirst integration allows you to build a Cordova app that uses the Cordova API and custom plug-ins (see "Creating a new Cordova app without the MobileFirst template" on page 7-89). The resulting app includes a seamless integration and you can develop the WebView in various IDEs that support JavaScript (see "Editing WebView (JavaScript) code" on page 7-129). Or you can view your project setup in Android Studio and customize the setup, add splash screens, configure the app or use the emulators to view the app.

Cordova offers a default WebView for JavaScript development or the Crosswalk WebView based on the Chrome browser. After you apply the Crosswalk plug-in to your Cordova app (see "Crosswalk WebView (Android)" on page 7-133) the Crosswalk WebView is used without requiring any further configuration.

The following topics show briefly the Cordova app setup for Android.

Running a Cordova Project in Android Studio:

Develop your Cordova app with MobileFirst for Android using Android Studio.

Before you begin

You must have a Cordova app project set up with Apache Cordova CLI (see "Creating a new Cordova app without the MobileFirst template" on page 7-89).

Procedure

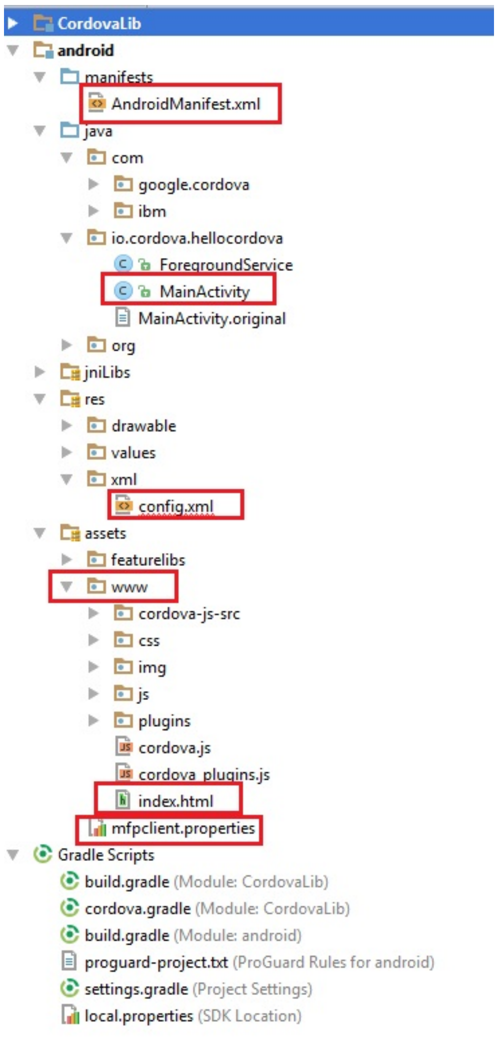
1. If you have not already done so, from the command line run

   ```
   cordova prepare
   ```

   This creates the `mfpclient.properties` file in the assets folder. If this file does not exist, you cannot run your app.
2. Open the project in Android Studio.
   a. From Android Studio select **File->New->Import Project**.
   b. Navigate to the `build.gradle` file in project name and click **OK**.
3. In the project navigator pane the project looks like this:

Table 7-15. Android Studio navigation pane

Android Studio navigation pane	Description of selected files

	`AndroidManifest.xml`
	
	One of the functions of the AndroidManifest.xml file is list the start up code for the app.
	
	`MainActivity.java`
	
	The start up code for the app.
	
	`config.xml`
	
	Various configurations including:
	• Calls hooks.
	• Points to the default html page: `<content src="index.html" />`
	
	`www`
	
	Contains the JavaScript, HTML, and CSS files loaded into the `WebView`.
	
	`index.html`
	
	The initial html page loaded by the WebView, defined in the `config.xml` file.
	
	`mfpclient.properties`
	
	The file containing information for connecting to the MobileFirst server. See

```
▶ CordovaLib
▼ android
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com
      ▶ google.cordova
      ▶ ibm
    ▼ io.cordova.hellocordova
        ForegroundService
        MainActivity
        MainActivity.original
    ▶ org
  ▶ jniLibs
  ▼ res
    ▶ drawable
    ▶ values
    ▼ xml
        config.xml
  ▼ assets
    ▶ featurelibs
    ▼ www
      ▶ cordova-js-src
      ▶ css
      ▶ img
      ▶ js
      ▶ plugins
        cordova.js
        cordova_plugins.js
        index.html
      mfpclient.properties
▼ Gradle Scripts
    build.gradle (Module: CordovaLib)
    cordova.gradle (Module: CordovaLib)
    build.gradle (Module: android)
    proguard-project.txt (ProGuard Rules for android)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

4. Run your application to view it in the emulator.
   a. From the **Run** menu choose **Run Android**.
   b. Choose a device from the **Device Chooser** dialog.

The app is displayed in the emulator.

What to do next

You now have your Cordova project set up in Android Studio. You must register your app before you can start writing your code. See "Registering Cordova applications to MobileFirst Server" on page 7-106.

Cordova application with MobileFirst start-up flow:

In Android Studio, you can review the start-up process of the Cordova app for Android with MobileFirst.

The MobileFirst Cordova plug-in `cordova-plugin-mfp` has native asynchronous bootstrap sequence. The bootstrap sequence must be completed before the Cordova application loads the application's main html file.

Adding the `cordova-plugin-mfp` plug-in to a Cordova application instruments the application's `AndroidManifest` file and the `MainActivity` extending the `CordovaActivity` native code to perform the MobileFirst initialization. MobileFirst can also be used in Cordova applications configured with the Crosswalk WebView.

The application native code instrumentation consists of:
- Adding `com.worklight.androidgap.api.WL` API calls to perform the MobileFirst initialization.
- In the `AndroidManifest.xml` file adding
  - An activity called `MFPLoadUrlActivity` to allow proper MobileFirst initialization in case the `cordova-plugin-crosswalk-webview` has been installed.
  - A custom attribute `android:name="com.ibm.MFPApplication"` to the `<application>` element (see below).

Implementing `WLInitWebFrameworkListener` and creating the `WL` object

The `MainActivity.java` file creates the initial `MainActivity` class extending the `CordovaActivity` class. The `WLInitWebFrameworkListener` receives notification when the MobileFirst framework is initialized.

```
public class MainActivity extends CordovaActivity implements WLInitWebFrameworkListener {
```

The `MFPApplication` class is called from within `onCreate` and creates a MobileFirst client instance (`com.worklight.androidgap.api.WL`) that is used throughout the app. The `onCreate` method initializes the WebView framework.

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

if (!((MFPApplication)this.getApplication()).hasCordovaSplashscreen()) {
          WL.getInstance().showSplashScreen(this);
      }
   init();
   WL.getInstance().initializeWebFramework(getApplicationContext(), this);
}
```

The `MFPApplication` class has two functions:
- Defines the `showSplashScreen` method for loading a splash screen if one exists. For more information on controlling splash screens see .

- Creates two listeners for enabling analytics. These listeners can be removed if not needed. For more information on Cordova and MobileFirst analytics see "MobileFirst Cordova plug-in initialization for analytics."

Loading the WebView

The `cordova-plugin-mfp` plug-in adds an activity to the `AndroidManifest.xml` file that is required for initializing the Crosswalks WebView:

`<activity android:name="com.ibm.MFPLoadUrlActivity" />`.

This activity is used to ensure the asynchronous initialization of the Crosswalk WebView as follows:

After the MobileFirst framework is initialized and ready to load in the WebView, the `onInitWebFrameworkComplete` connects to the URL if `WLInitWebFrameworkResult` succeeds.

```
public void onInitWebFrameworkComplete(WLInitWebFrameworkResult result){
if (result.getStatusCode() == WLInitWebFrameworkResult.SUCCESS) {
super.loadUrl(WL.getInstance().getMainHtmlFilePath());
   } else {
      handleWebFrameworkInitFailure(result);
   }
}
```

For details on the WebView development see "Cordova WebView" on page 7-129.

MobileFirst Cordova plug-in initialization for analytics:

The MobileFirst Cordova plug-in prepares your app for sending analytics data to the server. If your app does not use analytics, this code can be removed.

After startup create two listeners for receiving device events. In the WebView JavaScript code, the data can then be sent with `WL.Analytics.send()`.

```
WLAnalytics.init(this);
WLAnalytics.addDeviceEventListener(WLAnalytics.DeviceEvent.NETWORK);
WLAnalytics.addDeviceEventListener(WLAnalytics.DeviceEvent.LIFECYCLE);
```

The MobileFirst Cordova plug-in inserts this listener code into the `onCreate` method of `MFPApplication`. If not needed it can be removed.

The `WL.Analytics.send()` must be added as needed in the `index.js` file, or any subsequently added JavasScript files.

For more information on MobileFirst see "Analytics and Logger" on page 11-1.

Developing Cordova apps for iOS

Details relevant to developing Cordova apps for iOS with MobileFirst are provided here.

Note: MobileFirst development is supported in Xcode from version 7.1 by using iOS 8.0 and later.

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

Cordova iOS applications with MobileFirst start-up flow:

The MobileFirst framework is initialized in the iOS platform to display a WebView in the Cordova app with MobileFirst.

main.m

In the `main.m` file the MobileFirst plug-in replaces the default main application `AppDelegate` with `MFPAppDelegate`.

```
#import <UIKit/UIKit.h>
int main(int argc, char *argv[]) {
 @autoreleasepool
          {
             int retVal = UIApplicationMain(argc, argv, nil, @"MFPAppDelegate");
             return retVal;
           }
}
```

MFPAppDelegate.m

The `MFPAppDelegate.m` file is found in the `plugins` folder. This replaces the default Cordova `AppDelegate.m` file and initializes the MobileFirst framework before the view controller loads the WebView .

The `didFinishLaunchingWithOptions` method initializes the framework:

```
[[WL sharedInstance] initializeWebFrameworkWithDelegate:self];
```

Once the initialization succeeds the `wlInitWebFrameworkDidCompleteWithResult` checks that the MobileFirst framework has been loaded, invokes `wlInitDidCompleteSuccessfully` and creates listeners for receiving data (see "MobileFirst Cordova plug-in initialization for analytics" on page 7-124. The `wlInitDidCompleteSuccessfully` creates a `cordovaViewController` that connects to the default `index.html` page.

Once the iOS Cordova app is built in Xcode without errors, you can proceed to add features to the native platform and WebView.

MobileFirst Cordova plug-in initialization of analytics:

The MobileFirst Cordova plug-in prepares your iOS app for sending analytics data to the server. If your app does not require analytics, this code can be removed.

MFPAppDelegate.m

The MobileFirst plug-in adds two lines for creating the listeners after the web framework initialization is complete and web resources are ready to be used. These appear in the `wlInitWebFrameworkDidCompleteWithResult` function in the `MFPAppDelegate.m` file.

```
[[WLAnalytics sharedInstance] addDeviceEventListener:NETWORK];
[[WLAnalytics sharedInstance] addDeviceEventListener:LIFECYCLE];
```

These lines can be removed if not needed.

To send analytics data to the server, the developer must explicitly add the call within `index.js` or any subsequently added JavaScript files:

```
WL.Analytics.send()
```

For more information on MobileFirst analytics see "MobileFirst Cordova plug-in initialization for analytics" on page 7-124.

Enabling OpenSSL for Cordova iOS:

The MobileFirst iOS SDK uses native iOS APIs for cryptography. You can configure the MobileFirst V8.0.0 to use the OpenSSL cryptography library in your Cordova iOS app.

The **encryption/decryption** functionalities are provided with the following Javascript APIs:

```
WL.SecurityUtils.encryptText
```

```
WL.SecurityUtils.decryptWithKey
```

Option 1: Native encryption/decryption

By default MobileFirst provides native encryption/decryption, without using OpenSSL. This is equivalent to explicitly setting the encryption/decryption behavior:

```
WL.SecurityUtils.enableNativeEncryption(true)
```

Option 2: Enabling OpenSSL

MobileFirst provided OpenSSL is disabled by default.

To install the necessary frameworks for supporting OpenSSL, first install the Cordova plug-in:

```
cordova plugin add cordova-plugin-mfp-encrypt-utils
```

The following code enables the **OpenSSL option** for the encryption/decryption:

```
WL.SecurityUtils.enableNativeEncryption(false)
```

With this setup, the encryption/decryption calls use OpenSSL as in previous versions of MobileFirst.

Migration options

If you have an IBM MobileFirst Platform Foundation project written in an earlier version, you may need to incorporate changes to continue using OpenSSL.
- If the application is not using encryption/decryption APIs, and no encrypted data is cached on the device, no action is needed.
- If the application is using encryption/decryption APIs you have the option of using these APIs with or without OpenSSL.
  - **Migrating to native encryption:**
    1. Make sure the default native encryption/decryption option is chosen (see **Option 1**).
    2. **Migrating cached data:** If the previous installation of IBM MobileFirst Platform Foundation saved encrypted data to the device using OpenSSL, but the native encryption/decryption option is now chosen, the stored data must be decrypted. The first time the application attempts to decrypt

the data it will fall back to OpenSSL and then encrypt it using native encryption. This way the data will be auto-migrated to native encryption.

Note: To allow the decryption from OpenSSL, you must add the OpenSSL frameworks by installing the Cordova plug-in:

```
cordova plugin add cordova-plugin-mfp-encrypt-utils
```

– **Continuing with OpenSSL:** If OpenSSL is required use the setup described in **Option 2**.

Enforcing TLS-secure connections for Cordova iOS:

From iOS 9 Transport Layer Security (TLS) protocol version 1.2 must be enforced in all iOS 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.

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>
```

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.

Developing Cordova apps for Windows

The Cordova app can run on the Windows platform. The MobileFirst integration allows you to build a Cordova app that uses the Cordova API and custom plug-ins.

About this task

To use MobileFirst to build a Cordova app that makes use of the Cordova API and custom plug-ins, see "Creating a new Cordova app without the MobileFirst template" on page 7-89. The resulting app includes a seamless integration and you can develop the WebView in various IDEs that support JavaScript. See "Editing WebView (JavaScript) code" on page 7-129. Or you can also choose to view your project setup in Visual Studio and customize the setup, add splash screens, configure the app or use the emulators to view the app.

The following topics helps you with the Cordova app setup for Windows.

Viewing a Cordova Project in Windows Visual Studio:

MobileFirst supports three different types of Windows Universal environments - Windows 8.1 Desktop, Windows Phone 8.1, and Windows 10 Universal Windows Platform (UWP).

About this task

Ensure that you have a Cordova app set up with Apache Cordova CLI. See "Creating a new Cordova app without the MobileFirst template" on page 7-89.

Note: For Windows 10 UWP apps, you need to have Visual Studio 2015.

Procedure

1. Navigate to the directory `<`*`Cordova_application_name`*`>/platforms/windows`.
2. Choose between either of the following methods to open your project.
   - Double click to open the solution file, `cordovaApp.sln`.
     – To work with a Windows project, choose the appropriate version in the Solution Explorer pane. You can choose between `Windows 8.1 Desktop`, `Windows Phone 8.1` or `Windows 10 UWP`. Right-click the required project name and select, **Set as StartUp Project**.
   - Choose to click the appropriate `.jsproj` file to open the project. The following versions of Windows are supported:
     – For Windows 8.1 Desktop, `Cordovaapp.Windows.jsproj`
     – For Windows Phone 8.1, `Cordovaapp.Phone.jsproj`
     – For Windows 10 UWP, `CordovaApp.Windows10.jsproj`

   The following folders and files are part of the project:
   - The `.appxmanifest` file contains the app name and info, as well as package information:

  - For Windows Phone 8.1: `package.phone.appxmanifest`
  - For Windows 8.1 Desktop: `package.windows.appxmanifest`
  - For Windows 10 UWP: `package.windows10.appxmanifest`
- The `config.xml` file contains various configurations, that includes the Calls hooks. The `config.xml` also points to the default html page: `<content src="index.html" />`.
- The `www` folder contains the JavaScript, HTML, and CSS files loaded into the WebView.
- The `index.html` is present in the `www` folder. It is the initial HTML page loaded by the WebView, defined in the config.xml file.
- The `mfpclient.properties` file contains information for connecting to the MobileFirst server.

Cordova application with MobileFirst start-up flow:

In Visual Studio, you can review the start-up process of the Cordova app for Windows with MobileFirst.

About this task

The MobileFirst Cordova plug-in, `cordova-plugin-mfp` has native asynchronous bootstrap sequence. The bootstrap sequence must be completed before the Cordova application loads the application's main HTML file.

Adding the `cordova-plugin-mfp` plug-in to a Cordova application adds the `index.html` file to the application's `appxmanifest` file. This extends the `CordovaActivity` native code to perform the MobileFirst initialization.

Cordova WebView

Once the native platform (iOS, Android or Windows) initializes the web framework, you can access MobileFirst functionality from the WebView with JavaScript.

You can use Cordova plug-ins or HTML 5 to create JavaScript API to call user-interface controls that are common to most environments, such as modal pop-up windows, loading screens, or tab bars.

From MobileFirst 8.0 you can integrate Crosswalk WebView with your Cordova MobileFirst app. See "Crosswalk WebView (Android)" on page 7-133.

Whether you choose to integrate Crosswalk or use the default Cordova WebView a wide range of MobileFirst functionality is available using JavaScript. See "Some initial WebView code for connecting to the server" on page 7-130.

Editing WebView (JavaScript) code:

Editing the WebView resources is more convenient using an IDE that provides autocompletion for JavaScript.

Xcode, Android Studio, and Visual Studio provide full editing capabilities for editing Objective C, Swift, C#, and Java, however they may be limited in how they assist the editing of JavaScript. To facilitate JavaScript editing, the MobileFirst Cordova project contains a defintion file for providing autocomplete for MobileFirst API elements.

Each MobileFirst Cordova plug-in provides a `d.ts` configuration file for each MobileFirst JavaScript files. The `d.ts` file name matches the corresponding JavaScript file name and is located within the plug-in folder. For example for the main MobileFirst SDK the file is here:

```
[myapp]\plugins\cordova-plugin-mfp\typings\worklight.d.ts
```

This definition provides autocomplete for all IDEs with TypeScript support:

TypeScript Playground

Visual Studio Code

WebStorm

WebEssentials

Eclipse

The resources (HTML and JavaScript files) for the WebView are located in the `[myapp]\www` folder. When the project is built with the `cordova build` command, or the `cordova prepare` command is run, these resources are copied to the corresponding `www` folder in the `[myapp]\platforms\ios\www`, `[myapp]\platforms\android\assets\www`, or `myapp]\platforms\windows\www` folder.

When you open the main app folder with one of the previous IDEs, the context is preserved. The IDE editor will now be linked to the relevant `d.ts` files and autocomplete the MobileFirst API elements as you type.

Some initial WebView code for connecting to the server:

Most of your Cordova app is developed using the web resources. After creating your Cordova app and adding the MobileFirst plug-in you can add MobileFirst functionality.

WebView files

In this example a simple `index.html` file creates the initial screen containing a single button. This button is defined in the imported `index.js` file.

Note: When running the Cordova CLI `build` or `prepare` command, the web resources in main `[projectname]\www` folder are copied to each platform's `www` folder (`[projectname]\platforms\[platform]\assets\www`). Therefore in this case you need to edit the main `www` copy if you are going to build with the Cordova command. If you are going to build with platform IDE such as Android Studio or Xcode, you need to edit the local platform copy.

After creating your Cordova app using the Apache tools and adding the MobileFirst plug-in, the `index.html` and JavaScript files do not include any examples of the MobileFirst API. Here is an example of accessing a server resource (adapter).

Note: You need to register your app before you can access the server. For more information, see "Registering Cordova applications to MobileFirst Server" on page 7-106.

The index.html file

The app starts up with the initial index.html The button is included in the html:

```
<div><button id ="resourceRequestBTN" style="float:left; margin-top:10px;">Resource Request</button> </div>
```

```
<html>
    <head>
        <meta http-equiv="Content-Security-Policy" content="default-src 'self' data: gap: https://ssl.gstatic.com 'unsafe-eval'; style-src 's
        <meta name="format-detection" content="telephone=no">
        <meta name="msapplication-tap-highlight" content="no">
        <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width">
        <link rel="stylesheet" type="text/css" href="css/index.css">
        <title>Hello World</title>
    </head>
    <body>
        <div class="app">
            <h1>Apache Cordova</h1>
            <div id="deviceready" class="blink">
                <p class="event listening">Connecting to Device</p>
                <p class="event received">Device is Ready</p>
                <br />
            </div>
            <div><button id ="resourceRequestBTN" style="float:left; margin-top:10px;">Resource Request</button> </div>
        </div>
        <script type="text/javascript" src="cordova.js"></script>
        <script type="text/javascript" src="js/index.js"></script>
    </body>
</html>
```

The resourceRequestBTN button is defined in the index.js file.

The index.js file

The index.js adds the listener to the resourceRequestBTN button which calls the resourceRequestGetBalance function.

```
document.getElementById("resourceRequestBTN").addEventListener("click", resourceRequestGetBalance, false);
```

The resourceRequestGetBalance function requests the server resource (an adapter) and returns either the response or an error.

```
function resourceRequestGetBalance(){
try{
        var request = new WLResourceRequest('/adapters/account/balance', WLResourceRequest.GET);
  request.send().then(
       function(response) {
           var response = JSON.stringify(response);
         console.log("response " + response);
         alert(response);
          // success flow, the result can be found in response.responseJSON
       },
       function(error) {
              console.log("error " + error);
          // failure flow
          // the error code and description can be found in error.errorCode and error.errorMsg fields respectively
       }
   ).fail(function(){
    console.log("WLResourceRequest failure");
   });
 }catch(err){
  console.log("exception:"+ err);
 }
}
```

For information on creating and deploying the adapter see "Developing the server side of a MobileFirst application" on page 7-187.

Testing the connection to the server without deploying an adapter

If you want to test the app registration and connection to the server without the need for deploying an adapter, you can test a request for an access token. If no security checks have been added to your app the access token will succeed once the registration and connectivity to the server are working.

To test the security token access replace the resource button with this button in the file:

```
<div id="btn1">   <button id ="getToken">getToken</button>  </div>
```

In the `index.js` file replace the resource button listener with the access token button listener for invoking `getToken`:

```
 document.getElementById("getToken").addEventListener("click", getToken, false);
```

and add the getToken function

```
function getToken() {
    WLAuthorizationManager.obtainAccessToken().then(function(token) {
        alert("success: "+JSON.stringify(token))
    },function(error) {
        alert("failure: "+JSON.stringify(error))
    });
}
```

If the connection succeeds you will see the contents of the token. If not the error is displayed.

The sample `index.js` file for the resource request check:

The entire `index.js` file for testing the resource request appears below.

```
var app = {
    // Application Constructor
    initialize: function() {
        this.bindEvents();
    },
    // Bind Event Listeners
    //
    // Bind any events that are required on startup. Common events are:
    // 'load', 'deviceready', 'offline', and 'online'.
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
    },
    // deviceready Event Handler
    //
    // The scope of 'this' is the event. In order to call the 'receivedEvent'
    // function, we must explicitly call 'app.receivedEvent(...);'
    onDeviceReady: function() {
        app.receivedEvent('deviceready');
        document.getElementById("resourceRequestBTN").addEventListener("click", resourceRequestGetBalance, false);
    },
    // Update DOM on a Received Event
    receivedEvent: function(id) {
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector('.listening');
        var receivedElement = parentElement.querySelector('.received');

        listeningElement.setAttribute('style', 'display:none;');
        receivedElement.setAttribute('style', 'display:block;');

        console.log('Received Event: ' + id);
    }
};

app.initialize();

function resourceRequestGetBalance(){
try{
        var request = new WLResourceRequest('/adapters/account/balance', WLResourceRequest.GET);
  request.send().then(
       function(response) {
           var response = JSON.stringify(response);
         console.log("response " + response);
         alert(response);
          // success flow, the result can be found in response.responseJSON
       },
       function(error) {
              console.log("error " + error);
          // failure flow
          // the error code and description can be found in error.errorCode and error.errorMsg fields respectively
       }
   ).fail(function(){
    console.log("WLResourceRequest failure");
   });
 }catch(err){
  console.log("exception:"+ err);
 }
}
```

Crosswalk WebView (Android):

You can replace the default WebView in your MobileFirst Cordova app with Crosswalk.

About this task

From MobileFirst V8.0.0 you can integrate Crosswalk WebView with your Cordova MobileFirst app.

Crosswalk provides an improved WebView using an internal browser based on Chrome.

To get started with Crosswalk install the Crosswalk plug-in to your Cordova project:

```
cordova plugin add cordova-plugin-crosswalk-webview
```

Once your app is created, your `res\xml\config.xml` file contains this line and the Crosswalk WebView is automatically chosen:

```
  <preference name="webView" value="org.crosswalk.engine.XWalkWebViewEngine" />
```

Note: The `cordova-plugin-crosswalk-webview` plug-in may use deprecated NDK integration. To allow Android Studio to build the application add a file name `gradle.properties` to the `platform\android` folder containing the following line:

```
android.useDeprecatedNdk=true
```

For more information about loading the MobileFirst Crosswalk WebView see "Loading the WebView" on page 7-124.

WKWebView (iOS):

You can replace the default UIWebView in your MobileFirst hybrid iOS Cordova app with WKWebView.

About this task

From IBM MobileFirst Platform Foundation V8.0.0, you can integrate WKWebView with your hybrid iOS Cordova MobileFirst app.

WKWebView provides an improved WebView. WKWebView displays interactive web content, such as for an in-app browser.

Procedure

To get started with WKWebView, install the WKWebView plug-in to your Cordova project.

```
cordova plugin add cordova-plugin-wkwebview-engine
```

Results

After your app is created, Cordova uses the WKWebView component instead of the default UIWebView component.

Note: To learn about known issues with WKWebView plug-in, see wkwebview-known-issues.

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, 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 Cordova applications, see `WL.JSONStore` in the API reference section. Cordova applications are supported for iOS, Android, Windows 10 Universal Windows Platform and Windows 8 Universal.

For JSONStore API reference information for native iOS applications, see the `JSONStore` Class Reference in the API reference section.

For JSONStore API reference information for native Android applications, see the `com.worklight.jsonstore.api` Package in the API reference section.

Here is a high-level summary of what JSONStore provides:

- A developer-friendly 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 7-171.

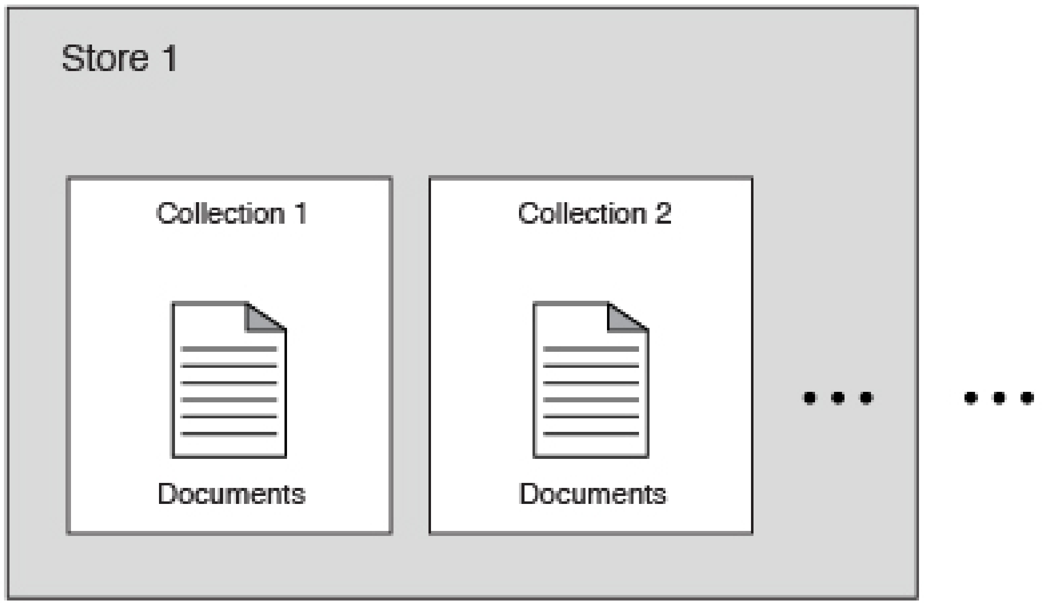


Figure 7-5. A basic graphic representation of JSONStore.

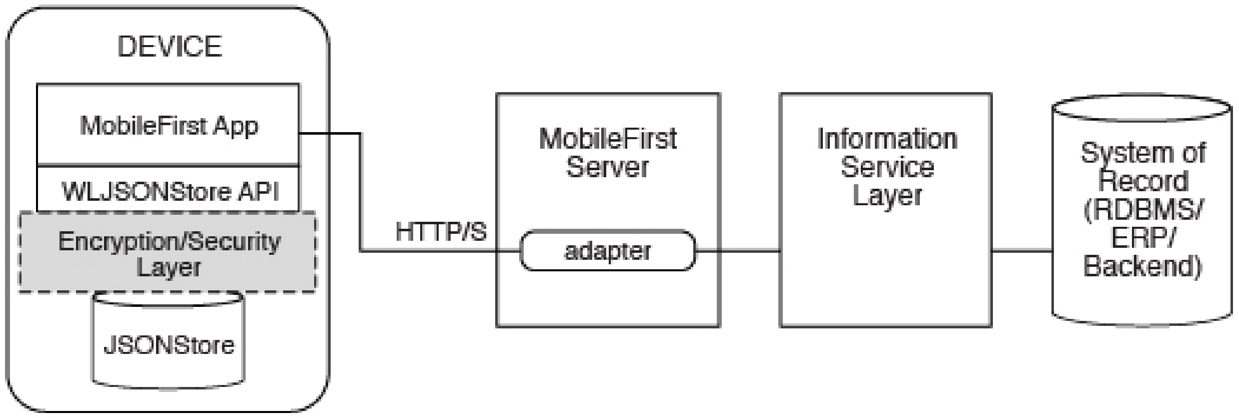

DEVICE
MobileFirst App
WLJSONStore API
Encryption/Security Layer
JSONStore
HTTP/S
MobileFirst Server
adapter
Information Service Layer
System of Record (RDBMS/ ERP/ Backend)

Figure 7-6. Components and their interaction with the server when you use JSONStore for data synchronization.

Note: Because it is familiar to developers, relational database terminology is used in this documentation at times to help explain JSONStore. There are many differences between a relational database and JSONStore however. For example, the strict schema that is used to store data in relational databases is different from JSONStore's approach. With JSONStore, you can store any JSON content, and index the content that you need to search.

Features table

Compare JSONStore features to those features of other data storage technologies and formats.

JSONStore is a JavaScript API for storing data inside Cordova applications that use the MobileFirst plug-in, an Objective-C API for native iOS applications, and a Java API for native Android applications. For reference, here is a comparison of different JavaScript storage technologies to see how JSONStore compares to them.

JSONStore is similar to technologies such as LocalStorage, Indexed DB, Cordova Storage API, and Cordova File API. The table shows how some features that are provided by JSONStore compare with other technologies. The JSONStore feature is only available on iOS and Android devices and simulators.

Table 7-16. A comparison of data storage technologies..

	JSONStore	**LocalStorage**	**IndexedDB**	**Cordova Storage**	**Cordova File**
Android Support (Cordova & Native Applications)	⌂	⌂	⌂	⌂	⌂
iOS Support (Cordova & Native Applications)	⌂	⌂	⌂	⌂	⌂
Windows 10 Universal Windows Platform and Windows 8 Universal (Cordova Applications)	⌂	⌂	⌂	-	⌂
Data encryption	⌂	-	-	-	-
Maximum Storage	Available Space	~5 MB	>5 MB	Available Space	Available Space
Reliable Storage (See Note 2)	⌂	-	-	⌂	⌂
Keep Track of Local Changes	⌂	-	-	-	-
Multi-user support	⌂	-	-	-	-
Indexing	⌂	-	⌂	⌂	-
Type of Storage	JSON Documents	Key/Value Pairs	JSON Documents	Relational (SQL)	Strings

Note: 2. *Reliable Storage* means that your data is not deleted unless one of the following events occurs:

- The application is removed from the device.
- One of the methods that removes data is called.

General JSONStore terminology

Learn about general JSONStore terminology.

JSONStore document

A document is the basic building block of JSONStore.

A JSONStore document is a JSON object with an automatically generated identifier (`_id`) and JSON data. It is similar to a record or a row in database terminology. The value of `_id` is always a unique integer inside a specific collection. Some functions like the `add`, `replace`, and `remove` methods in the JSONStoreInstance class take an Array of Documents/Objects. These methods are useful to perform operations on various Documents/Objects at a time.

Example

Single document

```
var doc = { _id: 1, json: {name: 'carlos', age: 99} };
```

Example

Array of documents

```
var docs = [
  { _id: 1, json: {name: 'carlos', age: 99} },
  { _id: 2, json: {name: 'tim', age: 100} }
]
```

JSONStore collection

A JSONStore collection is similar to a table, in database terminology

Example

Customer collection

```
[
    { _id: 1, json: {name: 'carlos', age: 99} },
    { _id: 2, json: {name: 'tim', age: 100} }
]
```

This code is not the way that the documents are stored on disk, but it is a good way to visualize what a collection looks like at a high level.

JSONStore store

A store is the persistent JSONStore file that contains one or more collections.

A store is similar to a relational database, in database terminology. A store is also referred to as a JSONStore.

JSONStore search fields

A search field is a 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 MobileFirst applications, you must take steps to enable it.

About this task

For iOS native apps, you must import the JSONStore iOS framework into your Xcode project.You can obtain the JSONStore feature in two ways:

- You can import the JSONStore framework manually or with CocoaPods. For more information, see "Setting up the Xcode project for iOS manually" on page 7-27.
- To use CocoaPods:
  - 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 named `Podfile` by using the **`pod init`** command.
     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:

        ```
        pod 'IBMMobileFirstPlatformFoundationJSONStore'
        ```

        Note: The above syntax imports assumes you are using the latest version of the `IBMMobileFirstPlatformFoundation`. 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 8.0.0 `IBMMobileFirstPlatformFoundation` the line would look like this:

        ```
        pod 'IBMMobileFirstPlatformFoundationJSONStore', '~> 8.0.0'
        ```
  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, called the `IBMMobileFirstPlatformFoundationJSONStore.framework` component, and integrates it with the mobile application Xcode project.
- You must import the `JSONStore.h` header file in your code to use the JSONStore API. For Objective C add #import `<IBMMobileFirstPlatformFoundationJSONStore/`

IBMMobileFirstPlatformFoundationJSONStore.h>. For Swift, add import IBMMobileFirstPlatformFoundationJSONStore.

For more information about creating native MobileFirst iOS applications, see "Developing native applications for iOS in Xcode" on page 7-27.

For native Android applications, you must add the files using gradle. For more information about creating native MobileFirst Android applications, see "Setting up Android Studio projects with Gradle" on page 7-53.

After adding the JSONStore SDK, you can use the classes inside the `com.worklight.jsonstore.api` package to use JSONStore.

For Cordova applications, you must add the `cordova-plugin-mfp-jsonstore` plug-in to your MobileFirst Cordova app. For more information see "Adding MobileFirst features to an existing Cordova app" on page 7-91.

JSONStore API concepts

JSONStore provides API reference information for Cordova Android, iOS, Windows 8 Universal, and native Android and iOS applications.

Store

Open and initialize a collection

Starts one or more collections. Starting or provisioning a JSONStore collection means that the persistent storage that is used to contain collections and documents is created, if it does not exist. If the store is encrypted and a correct password is passed, the required security procedures to make the data accessible are run. There is minimal effort in initializing all the collections when an application starts.

After you open a collection, an accessor to the collection is available, which gives access to collection APIs. It allows developers to call functions such as `find`, `add`, and `replace` on an initialized collection.

It is possible to initialize multiple times with different collections. New collections are initialized without affecting collections that are already initialized.

Destroy

Completely wipes data for all users, destroys the internal storage, and clears security artifacts. The `destroy` function removes the following data:

- All documents.
- All collections.
- All stores. For more information, see "JSONStore multiple user support" on page 7-171.
- All JSONStore metadata and security artifacts. For more information, see "JSONStore security" on page 7-169.

Close all

Locks access to all the collections in a store until the collections are reinitialized. Where `initialize` can be considered a login, `close` can be considered a logout.

Start, commit, and rollback transaction

A transaction is a set of operations that must all succeed for the operations to manipulate the store. If any operation fails, the transaction can be rolled back to revert the store to its previous state. After a transaction is started, it is important that you handle committing or rolling back your transactions to prevent excess processing. Three operations exist in the `Store` API for transactions:

- **Start transaction**
  Begin a snapshot in which the store is reverted to if the transaction fails.
- **Commit transaction**
  Inform the store that all operations in the transaction succeeded, and all changes can be finalized.
- **Rollback transaction**
  Inform the store that an operation in the transaction failed, and all changes must be discarded.

Note: Due to system limitations with multi-threaded transactions, transactions are not supported in Android 2.3.x for Cordova applications. To use transactions in a Cordova application in Android 2.3.x, you can create a Cordova plug-in that uses the native Android JSONStore API to execute the code for the transaction. The whole transaction must be done in the same thread because multi-threaded transactions do not work properly in Android 2.3.x.

Collection

Store and add a document

You can add a document or array of documents to a collection. You can also pass an array of objects (for example `[{name: 'carlos'}, {name: 'tim'}]`) instead of a single object. Every object in the array is stored as a new document inside the collection.

Remove a document

Marks one or more documents as removed from a collection. Removed documents are not returned by the `find` or `count` operations.

Find All Documents, Find Documents by Id, and Find With Query

You can find documents in a collection by their search fields and extra search fields. An internal search field, `_id`, holds a unique integer identifier that can be used to find the document (Find by Id). You can search for documents with the following APIs:

- **Find All Documents**
  Returns every document in a collection.
- **Find All Dirty Documents**
  Returns every document in a collection that is marked dirty.

- **Find by Id**
  Find the document with the corresponding `_id` search key value.
- **Find With Query or Query Parts**
  Find all documents that match a query or all query parts. For more information, see the Search Query format section at "Additional references" on page 7-143.

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 7-143.

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, Windows XP SP3 Virtual Machine or Mac OSX 10.8.3.
- Eclipse version. For example, Eclipse Indigo 3.7 Java EE.
- JDK version. For example, Java SE Runtime Environment (build 1.7).
- IBM MobileFirst Platform Foundation version. For example, IBM Worklight V5.0.6 Developer Edition.
- iOS version. For example, iOS Simulator 6.1 or iPhone 4S iOS 6.0 (deprecated, see "Deprecated features and API elements" on page 3-17).
- Android version. For example, Android Emulator 4.1.1 or Samsung Galaxy Android 4.0 API Level 14.
- Windows version. For example, Windows 8, Windows 8.1, or Windows Phone 8.1.
- adb version. For example, Android Debug Bridge version 1.0.31.
- Logs, such as Xcode output on iOS or logcat output on Android.

Try to isolate the issue

Follow these steps to isolate the issue to more accurately report a problem.

1. Reset the emulator (Android) or simulator (iOS) and call the `destroy` API to start with a clean system.
2. Ensure that you are running on a supported production environment.
   - Android >= 2.3 ARM v7/ARM v8/x86 emulator or device
   - iOS >= 6.0 simulator or device (deprecated)
   - Windows Phone Silverlight 8.0 ARM/x86 emulator or device
   - Windows 8.0-8.1 ARM/x86/x64 simulator or device

3. Try to turn off encryption by not passing a password to the `init` or `open` APIs.
4. Look at the SQLite database file that is generated by JSONStore. Encryption must be turned off.
   - Android emulator:
     ```
     $ adb shell
     $ cd /data/data/com.<app-name>/databases/wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   - iOS simulator:
     ```
     $ cd ~/Library/Application Support/iPhone Simulator/7.1/Applications/<id>/Documents/wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   - Windows Phone Silverlight 8:
     ```
     $ cd C:\Data\Users\DefApps\AppData\<id>\Local\wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   - Windows 8 Universal simulator
     ```
     $ cd C:\Users\<username>\AppData\Local\Packages\<id>\LocalState\wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   -

     Note: JavaScript only implementation that runs on a web browser (Firefox, Chrome, Safari, Internet Explorer) does not use an SQLite database. The file is stores in HTML5 LocalStorage.
   - Look at the `searchFields` with `.schema` and select data with `SELECT * FROM <collection-name>;`. To exit sqlite3, type `.exit`. If you pass a user name to the `init` method, the file is called *<username>*`.sqlite`. If you do not pass a user name, the file is called `jsonstore.sqlite` by default.
5. (Android only) Enable verbose JSONStore.
   ```
   adb shell setprop log.tag.jsonstore-core VERBOSE
   adb shell getprop log.tag.jsonstore-core
   ```
6. Use the debugger.

Common issues

Understanding the following JSONStore characteristics can help resolve some of the common issues that you might encounter.

- The only way to store binary data in JSONStore is to first encode it in `base64`. Store file names or paths instead of the actual files in JSONStore.
- Accessing JSONStore data from native code is possible only in IBM MobileFirst Platform Foundation V6.2.0.
- There is no limit on how much data you can store inside JSONStore, beyond limits that are imposed by the mobile operating system.
- JSONStore provides persistent data storage. It is not only stored in memory.
- The `init` API fails when the collection name starts with a digit or symbol. IBM Worklight V5.0.6.1 and later returns an appropriate error:
  ```
  4 BAD_PARAMETER_EXPECTED_ALPHANUMERIC_STRING
  ```
- There is a difference between a number and an integer in search fields. Numeric values like `1` and `2` are stored as `1.0` and `2.0` when the type is `number`. They are stored as `1` and `2` when the type is `integer`.
- If an application is forced to stop or crashes, it always fails with error code `-1` when the application is started again and the `init` or `open` API is called. If this problem happens, call the `closeAll` API first.
- The JavaScript implementation of JSONStore expects code to be called serially. Wait for an operation to finish before you call the next one.

- Transactions are not supported in Android 2.3.x for Cordova applications. For more information, see "JSONStore API concepts" on page 7-140.
- 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 7-17. 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 7-140 or "Work with external data" on page 7-174.

`_deleted`
Marks a document as deleted or not. This field is useful to remove objects from the collection, to later use them to track changes with your backend and decide whether to remove them or not.

`_operation`
A string that reflects the last operation to be performed on the document (for example, replace).

JSONStore errors:

Learn about JSONStore errors.

Possible JSONStore error codes that are returned are listed in "JSONStore error codes" on page 7-148.

JavaScript

JSONStore uses an error object to return messages about the cause of failures.

When an error occurs during a JSONStore operation (for example the `find`, and `add` methods in the JSONStoreInstance class) an error object is returned. It provides information about the cause of the failure.

Example

```
var errorObject = {
  src: 'find', // Operation that failed.
  err: -50, // Error code.
  msg: 'PERSISTENT_STORE_FAILURE', // Error message.
  col: 'people', // Collection name.
  usr: 'jsonstore', // User name.
  doc: {_id: 1, {name: 'carlos', age: 99}}, // Document that is related to the failure.
  res: {...} // Response from the server.
}
```

Not all the key/value pairs are part of every error object. For example, the doc value is only available when the operation failed because of a document (for example the `remove` method in the JSONStoreInstance class) failed to remove a document.

Objective-C

All of the APIs that might fail take an error parameter that takes an address to an NSError object. If you don not want to be notified of errors, you can pass in `nil`. When an operation fails, the address is populated with an NSError, which has an error and some potential `userInfo`. The `userInfo` might contain extra details (for example, the document that caused the failure).

Example

```
// This NSError points to an error if one occurs.
NSError* error = nil;

// Perform the destroy.
[JSONStore destroyDataAndReturnError:&error];
```

Java

All of the Java API calls throw a certain exception, depending on the error that happened. You can either handle each exception separately, or you can catch `JSONStoreException` as an umbrella for all JSONStore exceptions.

Example

```
try {
  WL.JSONStore.closeAll();
}

catch(JSONStoreException e) {
  // Handle error condition.
}
```

JSONStore error codes:

Definitions of the error codes that are related to JSONStore.

`-100 UNKNOWN_FAILURE`
Unrecognized error.

`-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.

`-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) or shared user preferences (Android).

-3 INVALID_KEY_ON_PROVISION
Passed the wrong password to an encrypted store.

-2 PROVISION_TABLE_SEARCH_FIELDS_MISMATCH
Search fields are not dynamic. It is not possible to change search fields without calling the `destroy` method or the `removeCollection` method before you call the `init` or `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 V5.0.6 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 : ⌂€œASC⌂€⌂}.

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 7-146.

30 `BAD_PARAMETER_EXPECTED_ARRAY_OF_OBJECTS`
Validation error when the array is not an array of only JSON objects.

31 `BAD_PARAMETER_EXPECTED_ARRAY_OF_CLEAN_DOCUMENTS`
Validation error.

32 `BAD_PARAMETER_WRONG_SEARCH_CRITERIA`
Validation error.

JSONStore examples

Learn about how to get started with JSONStore examples.

JavaScript API examples:

You can use JSONStore for Cordova applications that use the MobileFirst plug-in.

The following sections contain example implementations for JavaScript with JSONStore APIs. Other helpful topics include:

- “JSONStore overview” on page 7-134 - Learn about key concepts.
- “Enabling JSONStore” on page 7-139 - Learn how to enable JSONStore in different environments.
- “JSONStore API concepts” on page 7-140 - Learn about general information about the APIs that apply to all implementations of the JSONStore API.
- “Troubleshooting JSONStore” on page 7-144 - Learn how to debug and understand possible errors.

- "JSONStore advanced topics" on page 7-169 - Learn about security, multiple user support, performance, and concurrency.
- Class JSONStoreInstance - Learn about JSONStore APIs for JavaScript.
- "Work with external data" on page 7-174 - 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

```
var collectionName = 'people';

// Object that defines all the collections.
var collections = {

  // Object that defines the 'people' collection.
  people : {

    // Object that defines the Search Fields for the 'people' collection.
    searchFields : {name: 'string', age: 'integer'}
  }
};

// Optional options object.
var options = {

  // Optional username, default 'jsonstore'.
  username : 'carlos',

  // Optional password, default no password.
  password : '123',

  // Optional local key generation flag, default false.
  localKeyGen : false
};

WL.JSONStore.init(collections, options)

.then(function () {

  // Data to add, you probably want to get
  // this data from a network call (e.g. MobileFirst Adapter).
  var data = [{name: 'carlos', age: 10}];

  // Optional options for add.
  var addOptions = {

    // Mark data as dirty (true = yes, false = no), default true.
    markDirty: true
  };

  // Get an accessor to the people collection and add data.
  return WL.JSONStore.get(collectionName).add(data, addOptions);
})

.then(function (numberOfDocumentsAdded) {
  // Add was successful.
})

.fail(function (errorObject) {
   // Handle failure for any of the previous JSONStore operations (init, add).
});
```

Find - locate documents inside the Store

```
var collectionName = 'people';

// Find all documents that match the queries.
```

```
var queryPart1 = WL.JSONStore.QueryPart()
                   .equal('name', 'carlos')
                   .lessOrEqualThan('age', 10)

var options = {
  // Returns a maximum of 10 documents, default no limit.
  limit: 10,

  // Skip 0 documents, default no offset.
  offset: 0,

  // Search fields to return, default: ['_id', 'json'].
  filter: ['_id', 'json'],

  // How to sort the returned values, default no sort.
  sort: [{name: WL.constant.ASCENDING}, {age: WL.constant.DESCENDING}]
};

WL.JSONStore.get(collectionName)

// Alternatives:
// - findById(1, options) which locates documents by their _id field
// - findAll(options) which returns all documents
// - find({'name': 'carlos', age: 10}, options) which finds all documents
// that match the query.
.advancedFind([queryPart1], options)

.then(function (arrayResults) {
  // arrayResults = [{_id: 1, json: {name: 'carlos', age: 99}}]
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Replace - change the documents that are already stored inside a Collection

```
var collectionName = 'people';

// Documents will be located with their '_id' field
// and replaced with the data in the 'json' field.
var docs = [{_id: 1, json: {name: 'carlitos', age: 99}}];

var options = {

  // Mark data as dirty (true = yes, false = no), default true.
  markDirty: true
};

WL.JSONStore.get(collectionName)

.replace(docs, options)

.then(function (numberOfDocumentsReplaced) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Remove - delete all documents that match the query

```
var collectionName = 'people';

// Remove all documents that match the queries.
var queries = [{_id: 1}];
var options = {

  // Exact match (true) or fuzzy search (false), default fuzzy search.
  exact: true,

  // Mark data as dirty (true = yes, false = no), default true.
  markDirty: true
};

WL.JSONStore.get(collectionName)

.remove(queries, options)

.then(function (numberOfDocumentsRemoved) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Count - gets the total number of documents that match a query

```
var collectionName = 'people';

// Count all documents that match the query.
// The default query is '{}' which will
// count every document in the collection.
var query = {name: 'carlos'};
var options = {

  // Exact match (true) or fuzzy search (false), default fuzzy search.
  exact: true
};

WL.JSONStore.get(collectionName)

.count(query, options)

.then(function (numberOfDocumentsThatMatchedTheQuery) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
WL.JSONStore.destroy()

.then(function () {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Security - close access to all opened Collections for the current user

```
WL.JSONStore.closeAll()

.then(function () {
  // Handle success.
})
```

```
.fail(function (errorObject) {
  // Handle failure.
});
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hard-coded in the example for brevity.
var oldPassword = '123';
var newPassword = '456';

var clearPasswords = function () {
  oldPassword = null;
  newPassword = null;
};

// Default username if none is passed is: 'jsonstore'.
var username = 'carlos';

WL.JSONStore.changePassword(oldPassword, newPassword, username)

.then(function () {

  // Make sure you do not leave the password(s) in memory.
  clearPasswords();

  // Handle success.
})

.fail(function (errorObject) {

  // Make sure you do not leave the password(s) in memory.
  clearPasswords();

  // Handle failure.
});
```

Push - get all documents that are marked as dirty, send them to a MobileFirst adapter, and mark them clean

```
var collectionName = 'people';
var dirtyDocs;

WL.JSONStore.get(collectionName)

.getAllDirty()

.then(function (arrayOfDirtyDocuments) {
  // Handle getAllDirty success.

  dirtyDocs = arrayOfDirtyDocuments;

  var procedure = 'procedure-name-1';
  var adapter = 'adapter-name';

  var resource = new WLResourceRequest("adapters/" + adapter + "/" + procedure, WLResourceRequest.GET);
  resource.setQueryParameter('params', [dirtyDocs]);
  return resource.send();
})

.then(function (responseFromAdapter) {
  // Handle invokeProcedure success.

  // You may want to check the response from the adapter
  // and decide whether or not to mark documents as clean.
  return WL.JSONStore.get(collectionName).markClean(dirtyDocs);
})

.then(function () {
  // Handle markClean success.
})
```

```
.fail(function (errorObject) {
  // Handle failure.
});
```

Pull - get new data from a MobileFirst adapter

```
var collectionName = 'people';

var adapter = 'adapter-name';
var procedure = 'procedure-name-2';

var resource = new WLResourceRequest("adapters/" + adapter + "/" + procedure, WLResourceRequest.GET);

resource.send()

.then(function (responseFromAdapter) {
  // Handle invokeProcedure success.

  // The following example assumes that the adapter returns an arrayOfData,
  // (which is not returned by default),
  // as part of the invocationResult object,
  // with the data that you want to add to the collection.
  var data = responseFromAdapter.responseJSON

  // Example:
  // data = [{id: 1, ssn: '111-22-3333', name: 'carlos'}];

  var changeOptions = {

    // The following example assumes that 'id' and 'ssn' are search fields,
    // default will use all search fields
    // and are part of the data that is received.
    replaceCriteria : ['id', 'ssn'],

    // Data that does not exist in the Collection will be added, default false.
    addNew : true,

    // Mark data as dirty (true = yes, false = no), default false.
    markDirty : false
  };

  return WL.JSONStore.get(collectionName).change(data, changeOptions);
})

.then(function () {
  // Handle change success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Check whether a document is dirty

```
var collectionName = 'people';
var doc = {_id: 1, json: {name: 'carlitos', age: 99}};

WL.JSONStore.get(collectionName)

.isDirty(doc)

.then(function (isDocumentDirty) {
  // Handle success.

  // isDocumentDirty - true if dirty, false otherwise.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Check the number of dirty documents

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.countAllDirty()

.then(function (numberOfDirtyDocuments) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Remove a Collection

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.removeCollection()

.then(function () {
  // Handle success.

  // Note: You must call the 'init' API to re-use the empty collection.
  // See the 'clear' API if you just want to remove all data that is inside.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Clear all data that is inside a Collection

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.clear()

.then(function () {
  // Handle success.

  // Note: You might want to use the 'removeCollection' API
  // instead if you want to change the search fields.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
WL.JSONStore.startTransaction()

.then(function () {
  // Handle startTransaction success.
  // You can call every JSONStore API method except:
  // init, destroy, removeCollection, and closeAll.

  var data = [{name: 'carlos'}];

  return WL.JSONStore.get(collectionName).add(data);
})
```

```
.then(function () {

  var docs = [{_id: 1, json: {name: 'carlos'}}];

  return WL.JSONStore.get(collectionName).remove(docs);
})

.then(function () {

  return WL.JSONStore.commitTransaction();
})

.fail(function (errorObject) {
  // Handle failure for any of the previous JSONStore operation.
  //(startTransaction, add, remove).

  WL.JSONStore.rollbackTransaction()

  .then(function () {
    // Handle rollback success.
  })

  .fail(function () {
    // Handle rollback failure.
  })

});
```

Get file information

```
WL.JSONStore.fileInfo()
.then(function (res) {
  //res => [{isEncrypted : true, name : carlos, size : 3072}]
})

  .fail(function () {
  // Handle failure.
});
```

Search with like, rightLike, and leftLike

```
// Match all records that contain the search string on both sides.
// %searchString%
var arr1 = WL.JSONStore.QueryPart().like('name', 'ca');  // returns {name: 'carlos', age: 10}
var arr2 = WL.JSONStore.QueryPart().like('name', 'los');  // returns {name: 'carlos', age: 10}

// Match all records that contain the search string on the left side and anything on the right side.
// searchString%
var arr1 = WL.JSONStore.QueryPart().rightLike('name', 'ca');  // returns {name: 'carlos', age: 10}
var arr2 = WL.JSONStore.QueryPart().rightLike('name', 'los');  // returns nothing

// Match all records that contain the search string on the right side and anything on the left side.
// %searchString
var arr = WL.JSONStore.QueryPart().leftLike('name', 'ca');  // returns nothing
var arr2 = WL.JSONStore.QueryPart().leftLike('name', 'los');  // returns {name: 'carlos', age: 10}
```

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 7-134 - Learn about key concepts.

- "Enabling JSONStore" on page 7-139 - Learn how to enable JSONStore in different environments.
- "JSONStore API concepts" on page 7-140 - Learn about general information about the APIs that apply to all implementations of the JSONStore API.
- "Troubleshooting JSONStore" on page 7-144 - Learn how to debug and understand possible errors.
- "JSONStore advanced topics" on page 7-169 - 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 7-174 - 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}]
```

Java API examples:

You can use JSONStore for Cordova applications that use the MobileFirst plug-in.

The following sections contain example implementations for Android devices with JSONStore APIs. Other helpful topics include:

- "JSONStore overview" on page 7-134 - Learn about key concepts.

- "Enabling JSONStore" on page 7-139 - Learn how to enable JSONStore in different environments.
- "JSONStore API concepts" on page 7-140 - Learn about general information about the APIs that apply to all implementations of the JSONStore API.
- "Troubleshooting JSONStore" on page 7-144 - Learn how to debug and understand possible errors.
- "JSONStore advanced topics" on page 7-169 - Learn about security, multiple user support, performance, and concurrency.
- Package `com.worklight.jsonstore.api` - Learn about JSONStore APIs for Java.
- "Work with external data" on page 7-174 - 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

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
  // Create the collections object that will be initialized.
  JSONStoreCollection peopleCollection = new JSONStoreCollection("people");
  peopleCollection.setSearchField("name", SearchFieldType.STRING);
  peopleCollection.setSearchField("age", SearchFieldType.INTEGER);
  collections.add(peopleCollection);

  // Optional options object.
  JSONStoreInitOptions initOptions = new JSONStoreInitOptions();
  // Optional username, default 'jsonstore'.
  initOptions.setUsername("carlos");
  // Optional password, default no password.
  initOptions.setPassword("123");

  // Open the collection.

  WLJSONStore.getInstance(ctx).openCollections(collections, initOptions);

  // Add data to the collection.
  JSONObject newDocument = new JSONObject("{name: 'carlos', age: 10}");
  JSONStoreAddOptions addOptions = new JSONStoreAddOptions();
  addOptions.setMarkDirty(true);
  peopleCollection.addData(newDocument, addOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations (init, add).
  throw ex;
} catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
throw ex;
}
```

Initialize with a secure random token from the server

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

// Do an AsyncTask because networking cannot occur inside the activity.
AsyncTask<Context, Void, Void> aTask = new AsyncTask<Context, Void, Void>() {
  protected Void doInBackground(Context... params) {
    final Context context = params[0];

    // Create the request listener that will have the
    // onSuccess and onFailure callbacks:
    WLRequestListener listener = new WLRequestListener() {
      public void onFailure(WLFailResponse failureResponse) {
        // Handle Failure.
      }

      public void onSuccess(WLResponse response) {
        String secureRandom = response.getResponseText();

        try {
          List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
          // Create the collections object that will be initialized.
          JSONStoreCollection peopleCollection = new JSONStoreCollection("people");
          peopleCollection.setSearchField("name", SearchFieldType.STRING);
          peopleCollection.setSearchField("age", SearchFieldType.INTEGER);
          collections.add(peopleCollection);

          // Optional options object.
          JSONStoreInitOptions initOptions = new JSONStoreInitOptions();

          // Optional username, default 'jsonstore'.
          initOptions.setUsername("carlos");

          // Optional password, default no password.
          initOptions.setPassword("123");

          initOptions.setSecureRandom(secureRandom);

          // Open the collection.
          WLJSONStore.getInstance(context).openCollections(collections, initOptions);

          // Other JSONStore operations (e.g. add, remove, replace, etc.) go here.
        }
        catch (JSONStoreException ex) {
          // Handle failure for any of the previous JSONStore operations (init, add).
          ex.printStackTrace();        }
      }
    };

    // Get the secure random from the server:
    // The length of the random string, in bytes (maximum is 64 bytes).
    int byteLength = 32;
    SecurityUtils.getRandomStringFromServer(byteLength, context, listener);
    return null;
  }
};
aTask.execute(ctx);
```

Find - locate documents inside the Store

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  JSONStoreQueryParts findQuery = new JSONStoreQueryParts();
  JSONStoreQueryPart part = new JSONStoreQueryPart();
  part.addLike("name", "carlos");
  part.addLessThan("age", 99);
  findQuery.addQueryPart(part);

  // Add additional find options (optional).
  JSONStoreFindOptions findOptions = new JSONStoreFindOptions();

  // Returns a maximum of 10 documents, default no limit.
  findOptions.setLimit(10);
  // Skip 0 documents, default no offset.
  findOptions.setOffset(0);
  // Search fields to return, default: ['_id', 'json'].
  findOptions.addSearchFilter("_id");
  findOptions.addSearchFilter("json");

  // How to sort the returned values, default no sort.
  findOptions.sortBySearchFieldAscending("name");
  findOptions.sortBySeachFieldDescending("age");

  // Find documents that match the query.
  List<JSONObject> results = peopleCollection.findDocuments(findQuery, findOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Replace - change the documents that are already stored inside a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Documents will be located with their '_id' field
  //and replaced with the data in the 'json' field.
  JSONObject replaceDoc = new JSONObject("{_id: 1, json: {name: 'carlitos', age: 99}}");

  // Mark data as dirty (true = yes, false = no), default true.
  JSONStoreReplaceOptions replaceOptions = new JSONStoreReplaceOptions();
  replaceOptions.setMarkDirty(true);

  // Replace the document.
  peopleCollection.replaceDocument(replaceDoc, replaceOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Remove - delete all documents that match the query

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Documents will be located with their '_id' field.
  int id = 1;

  JSONStoreRemoveOptions removeOptions = new JSONStoreRemoveOptions();

  // Mark data as dirty (true = yes, false = no), default true.
  removeOptions.setMarkDirty(true);

  // Replace the document.
  peopleCollection.removeDocumentById(id, removeOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
  throw ex;
}
```

Count - gets the total number of documents that match a query

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  // Count all documents that match the query.
  JSONStoreQueryParts countQuery = new JSONStoreQueryParts();
  JSONStoreQueryPart part = new JSONStoreQueryPart();

  // Exact match.
  part.addEqual("name", "carlos");
  countQuery.addQueryPart(part);

  // Replace the document.
  int resultCount = peopleCollection.countDocuments(countQuery);
  JSONObject doc = peopleCollection.findDocumentById(resultCount);
  peopleCollection.replaceDocument(doc);
}
catch (JSONStoreException ex) {
  throw ex;
}
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Destroy the Store.
  WLJSONStore.getInstance(ctx).destroy();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Security - close access to all opened Collections for the current user

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Close access to all collections.
  WLJSONStore.getInstance(ctx).closeAll();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hard-coded in the example for brevity.
String username = "carlos";
String oldPassword = "123";
String newPassword = "456";

// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  WLJSONStore.getInstance(ctx).changePassword(oldPassword, newPassword, username);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
finally {
  // It is good practice to not leave passwords in memory
  oldPassword = null;
  newPassword = null;
}
```

Push - get all documents that are marked as dirty, send them to a MobileFirst adapter, and mark them clean

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Check if document with _id 3 is dirty.
  List<JSONObject> allDirtyDocuments = peopleCollection.findAllDirtyDocuments();

  // Handle the dirty documents here (e.g. calling an adapter).

  peopleCollection.markDocumentsClean(allDirtyDocuments);
}  catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Pull - get new data from a MobileFirst adapter

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Pull data here and place in newDocs. For this example, it is hard-coded.
  List<JSONObject> newDocs = new ArrayList<JSONObject>();
  JSONObject doc = new JSONObject("{id: 1, ssn: '111-22-3333', name: 'carlos'}");
  newDocs.add(doc);

  JSONStoreChangeOptions changeOptions = new JSONStoreChangeOptions();

  // Data that does not exist in the collection will be added, default false.
  changeOptions.setAddNew(true);

  // Mark data as dirty (true = yes, false = no), default false.
  changeOptions.setMarkDirty(true);

  // The following example assumes that 'id' and 'ssn' are search fields,
  // default will use all search fields
  // and are part of the data that is received.
  changeOptions.addSearchFieldToCriteria("id");
  changeOptions.addSearchFieldToCriteria("ssn");

  int changed = peopleCollection.changeData(newDocs, changeOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
  throw ex;
}
```

Check whether a document is dirty

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Check if document with id '3' is dirty.
  boolean isDirty = peopleCollection.isDocumentDirty(3);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Check the number of dirty documents

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Get the count of all dirty documents in the people collection.
  int totalDirty = peopleCollection.countAllDirtyDocuments();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Remove a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Remove the collection. The collection object is
  // no longer usable.
  peopleCollection.removeCollection();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Clear all data that is inside a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  // Clear the collection.
  peopleCollection.clearCollection();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people");

  WLJSONStore.getInstance(ctx).startTransaction();

  JSONObject docToAdd = new JSONObject("{name: 'carlos', age: 99}");
  // Find documents that match query.
  peopleCollection.addData(docToAdd);


  //Remove added doc.
  int id = 1;
  peopleCollection.removeDocumentById(id);

  WLJSONStore.getInstance(ctx).commitTransaction();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.

  // An exception occured. Take care of it to prevent further damage.
  WLJSONStore.getInstance(ctx).rollbackTransaction();

  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.

  // An exception occured. Take care of it to prevent further damage.
  WLJSONStore.getInstance(ctx).rollbackTransaction();

  throw ex;
}
```

Get file information

```
Context ctx = getContext();
List<JSONStoreFileInfo> allFileInfo = WLJSONStore.getInstance(ctx).getFileInfo();

for(JSONStoreFileInfo fileInfo : allFileInfo) {
  long fileSize = fileInfo.getFileSizeBytes();
  String username = fileInfo.getUsername();
  boolean isEncrypted = fileInfo.isEncrypted();
}
```

JSONStore advanced topics

Learn about JSONStore advanced topics.

JSONStore security:

You can secure all of the collections in a store by encrypting them.

To encrypt all of the collections in a store, pass a password to the `init` (JavaScript) or `open` (Native iOS and Native Android) API. If no password is passed, none of the documents in the store collections are encrypted.

Some security artifacts (for example *salt*) are stored in the keychain (iOS), shared preferences (Android), isolated storage (Windows 8 Phone), or the credential locker

(Windows 8). The store is encrypted with a 256-bit Advanced Encryption Standard (AES) key. All keys are strengthened with Password-Based Key Derivation Function 2 (PBKDF2).

Data encryption is only available on Android, iOS, Windows 8 Phone, and Windows 8 environments. You can choose to encrypt 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. This process is not applicable to Android because the encrypted DPK is stored in shared preferences and wiped out when the application is uninstalled.

The first time that JSONStore opens a collection with a password, which means that the developer wants to encrypt data inside the store, JSONStore needs a random token. That random token can be obtained from the client or from the server.

When the `localKeyGen` key is present in the JavaScript implementation of the JSONStore API, and it has a value of `true`, a cryptographically secure token is generated locally. Otherwise, the token is generated by contacting the server, thus requiring connectivity to the MobileFirst Server. This token is required only the first time that a store is opened with a password. The native implementations (Objective-C and Java) generate a cryptographically secure token locally by default, or you can pass one through the `secureRandom` option.

The trade-off is between opening a store offline and trusting the client to generate that random token (less secure), or opening the store with access to the MobileFirst Server (requires connectivity) and trusting the server (more secure).

Windows 8 Universal encryption:

You can secure all of the collections in a store by encrypting them.

JSONStore uses SQLCipher as its underlying database technology. SQLCipher is a build of SQLite that is produced by Zetetic, LLC adds a layer of encryption to the database.

JSONStore uses SQLCipher on all platforms. On Android and iOS a free, open source version of SQLCipher is available, known as the Community Edition and is incorporated into the versions of JSONStore that is included in IBM MobileFirst Platform Foundation. The Windows versions of SQLCipher are only available under a commercial license and cannot be directly redistributed by IBM MobileFirst Platform Foundation.

Instead, JSONStore for Windows 8 Universal include SQLite as the underlying database. If you need to encrypt data for either of these platforms, you need to acquire your own version of SQLCipher and swap out the SQLite version that is included in IBM MobileFirst Platform Foundation. For more information, see "SQLCipher on Windows 8 Universal."

If you do not need encryption, the JSONStore is fully functional (minus encryption) by using the SQLite version in IBM MobileFirst Platform Foundation.

SQLCipher on Windows 8 Universal:

To use JSONStore encryption on Windows, you must replace SQLite with SQLCipher.

Replacing SQLite with SQLCipher for Windows 8 Universal:

To use JSONStore encryption on Windows 8 Universal, you must replace SQLite with SQLCipher.

Procedure

1. Run the SQLCipher for Windows Runtime 8.1 extension that comes with the SQLCipher for Windows Runtime Commercial Edition.
2. After the extension finishes installing, locate the SQLCipher version of the `sqlite3.dll` file that was just created. There is one for x86, one for x64, and one for ARM.
   ```
   C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1\ExtensionSDKs\SQLCipher.WinRT81\3.0.1\Redist\Retail\<platform>
   ```
3. Copy and replace this file to your MobileFirst application.
   ```
   <Worklight project name>\apps\<application name>\windows8\native\buildtarget\<platform>
   ```

JSONStore multiple user support:

With JSONStore, you can create multiple stores that contain different collections in a single MobileFirst application.

The `init` (JavaScript) or `open` (Native iOS and Native Android) API can take an options object with a user name. Different stores are separate files in the file system. The user name is used as the file name of the store. These separate stores can be encrypted with different passwords for security and privacy reasons. Calling the `closeAll` API removes access to all the collections. It is also possible to change the password of an encrypted store by calling the `changePassword` API.

An example use case would be various employees that share a physical device (for example an iPad or Android tablet) and 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

- IBM MobileFirst Platform Foundation provides APIs for getting information about the network, for example, `WL.Device.getNetworkInfo` (JavaScript). 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

- When you use the JavaScript API, JSONStore documents are serialized and deserialized as Strings between the Native (Objective-C, Java, or C#) Layer and the JavaScript Layer. One way to mitigate possible memory issues is by using limit and offset when you use the `find` API. That way, you limit the amount of memory that is allocated for the results and can implement things like pagination (show X number of results per page).
- Instead of using long key names that are eventually serialized and deserialized as Strings, consider mapping those long key names into smaller ones (for example:`myVeryVeryVerLongKeyName` to `k` or `key`). Ideally, you map them to short key names when you send them from the adapter to the client, and map them to the original long key names when you send data back to the backend.
- Consider splitting the data inside a store into various collections. Have small documents over various collections instead of monolithic documents in a single collection. This consideration depends on how closely related the data is and the use cases for said data.
- When you use the `add` API with an array of objects, it is possible to run into memory issues. To mitigate this issue, call these methods with fewer JSON objects at a time.
- JavaScript and Java have garbage collectors, while Objective-C has Automatic Reference Counting. Allow it to work, but do not depend on it entirely. Try to null references that are no longer used and use profiling tools to check that memory usage is going down when you expect it to go down.

CPU

- The amount of search fields and extra search fields that are used affect performance when you call the `add` method, which does the indexing. Only index the values that are used in queries for the `find` method.
- By default, JSONStore tracks local changes to its documents. This behavior can be disabled, thus saving a few cycles, by setting the `markDirty` flag to `false` when you use the `add`, `remove`, and `replace` APIs.
- Enabling security adds some overhead to the `init` or `open` APIs and other operations that work with documents inside the collection. Consider whether security is genuinely required. For example, the `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.

JavaScript

Most of the operations that can be performed on a collection, such as `add` and `find`, are asynchronous. These operations return a jQuery promise that is resolved when the operation completes successfully and rejected when a failure occurs. These promises are similar to success and failure callbacks.

A jQuery Deferred is a promise that can be resolved or rejected. The following examples are not specific to JSONStore, but are intended to help you understand their usage in general.

The Options Object with `onSuccess` and `onFailure` callbacks that were used in JSONStore for IBM Worklight V5.0.5 are deprecated in favor of promises.

Instead of promises and callbacks, you can also listen to JSONStore success (`'WL/JSONSTORE/SUCCESS'`) and failure (`'WL/JSONSTORE/FAILURE'` events. Perform actions that are based on the arguments that are passed to the event listener.

Example promise definition

```
var asyncOperation = function () {
  // Assumes that you have jQuery defined via $ in the environment
  var deferred = $.Deferred();

  setTimeout(function() {
    deferred.resolve('Hello');
  }, 1000);

  return deferred.promise();
};
```

Example promise usage

```
// The function that is passed to .then is executed after 1000 ms.
asyncOperation.then(function (response) {
  // response = 'Hello'
});
```

Example callback definition

```
var asyncOperation = function (callback) {
  setTimeout(function() {
    callback('Hello');
  }, 1000);
};
```

Example callback usage

```
// The function that is passed to asyncOperation is executed after 1000 ms.
asyncOperation(function (response) {
  // response = 'Hello'
});
```

Example events

```
$(document.body).on('WL/JSONSTORE/SUCCESS', function (evt, data, src, collectionName) {

  // evt - Contains information about the event
  // data - Data that is sent ater the operation (add, find, etc.) finished
  // src - Name of the operation (add, find, push, etc.)
  // collectionName - Name of the collection
});
```

Objective-C

When you use the Native iOS API for JSONStore, all operations are added to a synchronous dispatch queue. This behavior ensures that operations that touch the store are executed in order on a thread that is not the main thread. For more information, see the Apple documentation at Grand Central Dispatch (GCD).

Java

When you use the Native Android API for JSONStore, all operations are executed on the main thread. You must create threads or use thread pools to have asynchronous behavior. All store operations are thread-safe.

Work with external data:

Learn about the different concepts that are required to work with external data.

For the actual API examples, see "JSONStore examples" on page 7-151.

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.

Example pull 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.

External Data Source: Backend REST endpoint

Imagine that you have a REST endpoint that read data from a database and returns it as an array of JSON objects.

```
app.get('/people', function (req, res) {

  var people = database.getAll('people');

  res.json(people);
});
```

The data that is returned can look like the following example:

```
[{id: 0, name: 'carlos', ssn: '111-22-3333'},
 {id: 1, name: 'mike', ssn: '111-44-3333'},
 {id: 2, name: 'dgonz' ssn: '111-55-3333')]
```

Transport Layer: MobileFirst adapter

Imagine that you created an adapter that is called `people` and you defined a procedure that is called `getPeople`. The procedure calls the REST endpoint and returns the array of JSON objects to the client. You might want to do more work here, for example, return only a subset of the data to the client.

```
function getPeople () {

  var input = {
    method : 'get',
    path : '/people'
  };

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

On the client, you can use the `WLResourceRequest` API to get the data. Additionally, you might want to pass some parameters from the client to the MobileFirst adapter. One example is a date with the last time that the client got new data from the external source through the MobileFirst adapter.

```
var adapter = 'people';
var procedure = 'getPeople';

var resource = new WLResourceRequest('/adapters' + '/' + adapter + '/' + procedure, WLResourceRequest.GET);
resource.send()
.then(function (responseFromAdapter) {
  // ...
});
```

Note: You might want to take advantage of the `compressResponse`, `timeout`, and other parameters that can be passed to the `WLResourceRequest` API.

Alternatively, you can skip the MobileFirst adapter and use something like jQuery.ajax to directly contact the REST endpoint with the data that you want to store.

```
$.ajax({
  type: 'GET',
  url: 'http://example.org/people',
})
.then(function (responseFromEndpoint) {
  // ...
});
```

Internal Data Source API: JSONStore

After you have the response from the backend, you can work with that data by using JSONStore.

JSONStore provides a way to track local changes. It enables some APIs to mark documents as dirty. The API records the last operation that was performed on the document, and when the document was marked as dirty. You can then use this information to implement features like data synchronization.

The change API takes the data and some options:

`replaceCriteria`
These search fields are part of the input data. They are used to locate documents that are already inside a collection. For example, if you select:

```
['id', 'ssn']
```

as the replace criteria, pass the following array as the input data:

```
[{id: 1, ssn: '111-22-3333', name: 'Carlos'}]
```

and the `people` collection already contains the following document:

```
{_id: 1,json: {id: 1, ssn: '111-22-3333', name: 'Carlitos'}}
```

The `change` operation locates a document that matches exactly the following query:

```
{id: 1, ssn: '111-22-3333'}
```

Then the `change` operation performs a replacement with the input data and the collection contains:

```
{_id: 1, json: {id:1, ssn: '111-22-3333', name: 'Carlos'}}
```

The name was changed from `Carlitos` to `Carlos`. If more than one document matches the replace criteria, then all documents that match are replaced with the respective input data.

`addNew`
When no documents match the replace criteria, the `change` API looks at the value of this flag. If the flag is set to `true`, the `change` API creates a new document and adds it to the store. Otherwise, no further action is taken.

`markDirty`
Determines whether the `change` API marks documents that are replaced or added as dirty.

An array of data is returned from the MobileFirst adapter:

```
.then(function (responseFromAdapter) {

  var accessor = WL.JSONStore.get('people');

  var data = responseFromAdapter.responseJSON;

  var changeOptions = {
    replaceCriteria : ['id', 'ssn'],
    addNew : true,
    markDirty : false
  };

  return accessor.change(data, changeOptions);
})

.then(function() {
  // ...
})
```

You can use other APIs to track changes to the local documents that are stored. Always get an accessor to the collection that you perform operations on.

```
var accessor = WL.JSONStore.get('people')
```

Then, you can add data (array of JSON objects) and decide whether you want it to be marked dirty or not. Typically, you want to set the `markDirty` flag to `false` when you get changes from the external source. Then, set the flag to `true` when you add data locally.

```
accessor.add(data, {markDirty: true})
```

You can also replace a document, and opt to mark the document with the replacements as dirty or not.

```
accessor.replace(doc, {markDirty: true})
```

Similarly, you can remove a document, and opt to mark the removal as dirty or not. Documents that are removed and marked dirty do not show up when you use the `find` API. However, they are still inside the collection until you use the `markClean` API, which physically removes the documents from the collection. If the document is not marked as dirty, it is physically removed from the collection.

```
accessor.remove(doc, {markDirty: true})
```

Push

Many systems use the term *push* to refer to sending data to an external source.

There are three important pieces:

Internal Data Source API
: This source is the JSONStore API that returns documents with local-only changes (dirty).

Transport Layer
: This source is how you want to contact the external data source to send the changes.

External Data Source
: This source is typically a database, REST or SOAP endpoint, among others, that receives the updates that the client made to the data.

Example push scenario

All of the following code examples are written in pseudocode that looks similar to JavaScript.

Note: Use 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) {
  var adapter = 'people',
  procedure = 'updatePeople';

  var resource = new WLResourceRequest('/adapters/' + adapter + '/' + procedure, WLResourceRequest.GET)
  resource.setQueryParameter('params', [dirtyDocs]);
  return resource.send();
})

.then(function (responseFromAdapter) {
  // ...
})
```

Note: You might want to take advantage of the `compressResponse`, `timeout`, and other parameters that can be passed to the `WLResourceRequest` 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 MFP.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 = [];
var adapter = 'people';
var procedure = 'addPerson';
var resource;

while (len--) {

  var currentDirtyDoc = dirtyDocs[len];

  switch (currentDirtyDoc._operation) {

    case 'add':
    case 'store':

    resource = new WLResourceRequest('/adapters/people/addPerson', WLResourceRequest.GET);
    resource.setQueryParameter('params', [currentDirtyDoc]);

      arrayOfPromises.push(resource.send());

    break;

    case 'replace':
    case 'refresh':

    resource = new WLResourceRequest('/adapters/people/replacePerson', WLResourceRequest.GET);
    resource.setQueryParameter('params', [currentDirtyDoc]);


      arrayOfPromises.push(resource.send());

    break;

    case 'remove':
    case 'erase':

    resource = new WLResourceRequest('/adapters/people/removePerson', WLResourceRequest.GET);
    resource.setQueryParameter('params', [currentDirtyDoc]);

      arrayOfPromises.push(resource.send());
  }
}

$.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 analytics:

You can enable the collection of analytics information for Android and iOS.

Overview

You can collect key pieces of analytics information that are related to JSONStore with the MobileFirst platform.

File information

File information is collected once per application session if the JSONStore API is called with the analytics flag set to `true`. An application session is defined as loading the application into memory and removing it from memory. You can use this information to determine how much space is being used by JSONStore content in the application.

Performance metrics

Performance metrics are collected every time a JSONStore API is called with information about the start and end times of an operation. You can use this information to determine how much time various operations take in milliseconds.

Examples

iOS

```
JSONStoreOpenOptions* options = [JSONStoreOpenOptions new];
[options setAnalytics:YES];

[[JSONStore sharedInstance] openCollections:@[...] withOptions:options error:nil];
```

Android

```
JSONStoreInitOptions initOptions = new JSONStoreInitOptions();
initOptions.setAnalytics(true);

WLJSONStore.getInstance(...).openCollections(..., initOptions);
```

JavaScript

This example applies only when the application is running on the Android or iOS environments.

```
var options = {
  analytics : true
};

WL.JSONStore.init(..., options);
```

JSONStore security utilities

Learn about JSONStore security utilities.

JSONStore security utilities overview:

The 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"
```

Android

```
import com.worklight.wlclient.api.SecurityUtils
```

JavaScript

No setup is required.

JSONStore security utilities examples:

Learn about JSONStore security utilities examples.

JSONStore security utilities iOS examples:

Learn about JSONStore security utilities iOS examples.

Encryption and decryption

```
// User provided password, hardcoded only for simplicity.
NSString* password = @"HelloPassword";

// Random salt with recommended length.
NSString* salt = [WLSecurityUtils generateRandomStringWithBytes:32];

// Recomended number of iterations.
int iterations = 10000;

// Populated with an error if one occurs.
NSError* error = nil;

// Call that generates the key.
NSString* key = [WLSecurityUtils generateKeyWithPassword:password
                                 andSalt:salt
                                 andIterations:iterations
                                 error:&error];

// Text that is encrypted.
NSString* originalString = @"My secret text";
NSDictionary* dict = [WLSecurityUtils encryptText:originalString
                                      withKey:key
                                      error:&error];
```

```
// Should return: 'My secret text'.
NSString* decryptedString = [WLSecurityUtils decryptWithKey:key
                                             andDictionary:dict
                                             error:&error];
```

Encode and decode base64

```
// Input string.
NSString* originalString = @"Hello world!";

// Encode to base64.
NSData* originalStringData = [originalString dataUsingEncoding:NSUTF8StringEncoding];
NSString* encodedString = [WLSecurityUtils base64StringFromData:originalStringData length:originalString.length];

// Should return: 'Hello world!'.
NSString* decodedString = [[NSString alloc] initWithData:[WLSecurityUtils base64DataFromString:encodedString] encoding: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];
```

JSONStore security utilities Android examples:

Learn about JSONStore security utilities Android examples.

Encryption and decryption

```
String password = "HelloPassword";
String salt = SecurityUtils.getRandomString(32);
int iterations = 10000;

String key = SecurityUtils.generateKey(password, salt, iterations);

String originalText = "Hello World!";

JSONObject encryptedObject = SecurityUtils.encrypt(key, originalText);

// Deciphered text will be the same as the original text.
String decipheredText = SecurityUtils.decrypt(key, encryptedObject);
```

Encode and decode base64

```
import android.util.Base64;

String originalText = "Hello World";
byte[] base64Encoded = Base64.encode(text.getBytes("UTF-8"), Base64.DEFAULT);

String encodedText = new String(base64Encoded, "UTF-8");

byte[] base64Decoded = Base64.decode(text.getBytes("UTF-8"), Base64.DEFAULT);

// Decoded text will be the same as the original text.
String decodedText = new String(base64Decoded, "UTF-8");
```

Get remote random

```
Context context; // This is the current Activity's context.
int byteLength = 32;

// Listener calls the callback functions after it gets the response from the server.
WLRequestListener listener = new WLRequestListener(){
  @Override
  public void onSuccess(WLResponse wlResponse) {
    // Implement the success handler.
  }

  @Override
  public void onFailure(WLFailResponse wlFailResponse) {
    // Implement the failure handler.
    }
};

SecurityUtils.getRandomStringFromServer(byteLength, context, listener);
```

Get local random

```
int byteLength = 32;
String randomString = SecurityUtils.getRandomString(byteLength);
```

JSONStore security utilities JavaScript examples:

Learn about JSONStore security utilities JavaScript examples.

Encryption and decryption

```
// Keep the key in a variable so that it can be passed to the encrypt and decrypt API.
var key;

// Generate a key.
WL.SecurityUtils.keygen({
  password: 'HelloPassword',
  salt: Math.random().toString(),
  iterations: 10000
})

.then(function (res) {

  // Update the key variable.
  key = res;

  // Encrypt text.
  return WL.SecurityUtils.encrypt({
    key: key,
    text: 'My secret text'
  });
})

.then(function (res) {

  // Append the key to the result object from encrypt.
  res.key = key;

  // Decrypt.
  return WL.SecurityUtils.decrypt(res);
})

.then(function (res) {

  // Remove the key from memory.
  key = null;

  //res => 'My secret text'
})

.fail(function (err) {
  // Handle failure in any of the previously called APIs.
});
```

Encode and decode base64

```
WL.SecurityUtils.base64Encode('Hello World!')
.then(function (res) {
  return WL.SecurityUtils.base64Decode(res);
})
.then(function (res) {
  //res => 'Hello World!'
})
.fail(function (err) {
  // Handle failure.
});
```

Get remote random

```
WL.SecurityUtils.remoteRandomString(32)
.then(function (res) {
  // res => deba58e9601d24380dce7dda85534c43f0b52c342ceb860390e15a638baecc7b
})
.fail(function (err) {
  // Handle failure.
});
```

Get local random

```
WL.SecurityUtils.localRandomString(32)
.then(function (res) {
  // res => 40617812588cf3ddc1d1ad0320a907a7b62ec0abee0cc8c0dc2de0e24392843c
})
.fail(function (err) {
  // Handle failure.
});
```

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 provides an API to enable certificate pinning. This API is supported in native iOS, native Android, and cross-platform Cordova 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 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.
- 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:
   - Native iOS: add the certificate to the application bundle
   - Native Android: place the certificate in the `assets` folder
   - Cross-platform (Cordova): place the certificate in the `*app-name*\www\certificates` folder

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 Android

Syntax:

```
public void pinTrustedCertificatePublicKey(String certificateFilename) throws IllegalArgumentException
```

Example:

```
WLClient.getInstance().pinTrustedCertificatePublicKey("myCertificate.cer");
```

The certificate pinning method will throw an exception in two cases:

- The file does not exist
- The file is in the wrong format

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

Cross-platform (Cordova)

Syntax:

```
WL.Client.pinTrustedCertificatePublicKey('certificateFilename').then(onSuccess,onFailure)
```

The certificate pinning method returns a promise:

- The certificate pinning method will call the `onSuccess` method in case of successful pinning.

- The certificate pinning method will trigger the `onFailure` callback in two cases:
  - The file does not exist
  - The file is in the wrong format

  Example:

  ```
  WL.Client.pinTrustedCertificatePublicKey('myCertificate.cer').then(onSuccess,onFailure)
  ```

  Later, if a secured request is made to a server whose certificate is not pinned, the `onFailure` callback of the specific request (for example, `WL.Client.connect` or `WLResourceRequest`) is called.

For more details on the certificate pinning API, see the following reference sections:
- Native Android: Java client-side API WLClient class.
- Native iOS: Objective-C client-side API WLClient class.
- Cross-platform (Cordova): JavaScript client-side API WL.Client class.

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.

Adapters overview

Adapters contain the server-side code of applications that are deployed on and serviced by IBM MobileFirst Platform Foundation.

MobileFirst adapters are deployed to a runtime in the IBM MobileFirst Platform Server and are used to securely connect client applications to enterprise back-end systems and cloud services. Adapters deliver data and perform any necessary server-side logic on this data.

Adapters support DevOps needs:
- 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.
- An adapter can have user-defined properties that can be configured by administration personnel, without redeploying the adapter. This feature lets you customize adapter behavior for different environments, for example development, testing, and production.

Adapters provide other benefits:
- **Universality**: Adapters support multiple integration technologies and back-end information systems.
- **Read-only and transactional capabilities**: Adapters support read-only and transactional access modes to back-end systems.
- **Rapid development and testing**: Use tools such as Apache Maven and MobileFirst Platform CLI. Adapters use simple XML syntax and are easily configured with the JavaScript API or Java API.
- **Security**: Adapters use an OAuth-based security framework that enables you to protect enterprise resources. There are built-in annotations that let you quickly define the scope for authorization permissions of the resources.

- **Transparency**: Data that is retrieved from back-end applications is exposed in a uniform manner, regardless of the adapter type. Application developers can access data uniformly, regardless of its source, format, and protocol.

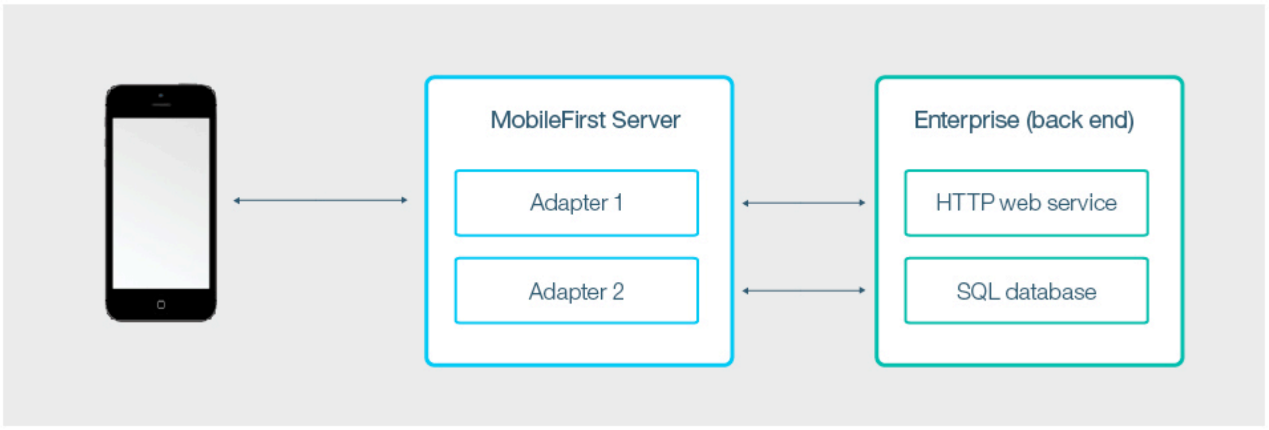


Figure 7-7. MobileFirst adapters

Adapters as Maven projects

Apache Maven is a popular tool for building and managing Java-based projects. Starting with IBM MobileFirst Platform Foundation V8.0.0, you can create, build, deploy, and configure a MobileFirst adapter as a Maven project.

For more information, see "Adapters as Apache Maven projects" on page 7-189.

Adapters and third-party dependencies

Every 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 by defining them as Maven dependencies in the adapter project `pom.xml` file. For more information, see "Third-party Maven dependencies" on page 7-192.

Note: There are certain limitations in the use of third-party dependencies in adapters. For more information, see "Adapters and third-party dependencies" on page 3-27.

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 calling other adapters, getting the user identity, and more. For more information, see "MobileFirst Java adapters" on page 7-192 and "MobileFirst JavaScript adapters" on page 7-204.

Lifecycle

The adapter lifecycle comprises four stages:

Develop
You develop your adapter to meet your needs. You can start by downloading a sample adapter from the Download Center in the IBM

MobileFirst Platform Operations Console, then develop it further using an IDE that supports Maven, such as Eclipse or IntelliJ.

- For more information about developing adapters with Eclipse or IntelliJ, see the following tutorials: Using IntelliJ to Develop MobileFirst Java Adapters and Developing Adapters in Eclipse.
- For information on additional facilities for developing adapters, see "Developing Java adapter code" on page 7-200 and "Developing JavaScript adapter code" on page 7-218.

Test and debug
MobileFirst Platform CLI makes it easy to test adapter code. It is also possible to use external tools such as Swagger, Postman or cURL to test the adapter. For more information, see "Tools for testing and debugging adapters" on page 7-230.

Deploy to staging and production environments
After development and testing have been completed, you deploy the adapter to a running MobileFirst Server.

- You can use the IBM MobileFirst Platform Operations Console to deploy the adapter. For more information, see "Deploying Java adapters" on page 7-198 and "Deploying JavaScript adapters" on page 7-216.
- It is also possible to automate the build and deploy process by using Apache Maven or Apache Ant deployer tasks. For more information, see "Administering MobileFirst applications through Ant" on page 10-23.

Configure
After developers have implemented the properties to be used in the adapter, administration staff can configure it on-the-fly, without having to redeploy it. For more information, see "Configuring adapters" on page 7-227.

Adapters as Apache Maven projects

When you use Maven projects to create, deploy, test, and configure adapters, you benefit from Maven's simple project setup, efficient dependency management, and supported IDEs, such as IntelliJ and Eclipse.

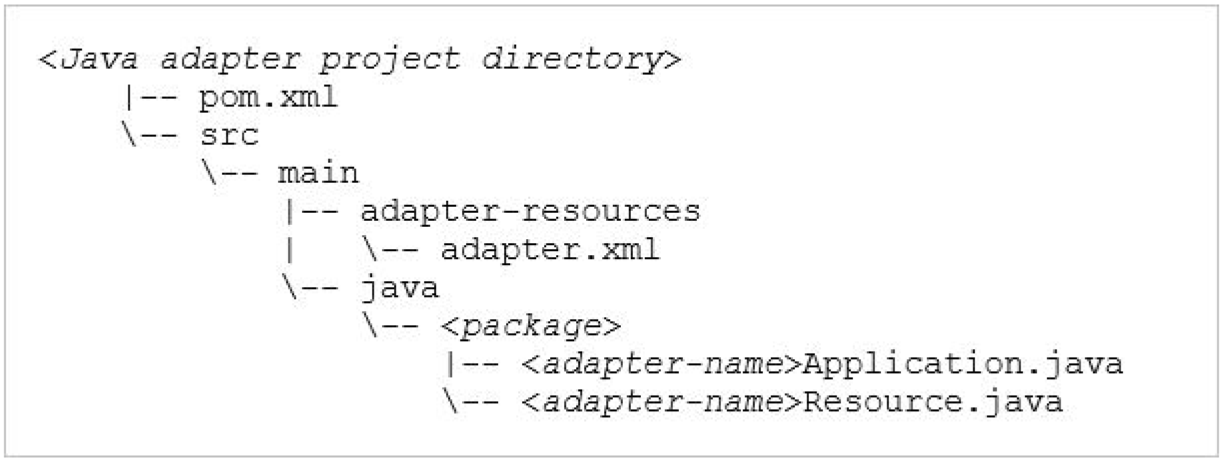
Maven repositories

Up-to-date Maven artifacts that support MobileFirst adapters are available at The Central Repository under the GroupId com.ibm.mfp. When you build a Maven project, Maven checks your `pom.xml` file to identify which dependency to download.

If your organization does not permit online access to The Central Maven Repository, your administrator must set up a local repository to hold and share MobileFirst artifacts. For more information, see "Setting up an internal Maven repository for offline development" on page 7-23.

Java adapter Maven projects

When you create a new Java adapter, a Maven project with the following structure is produced:



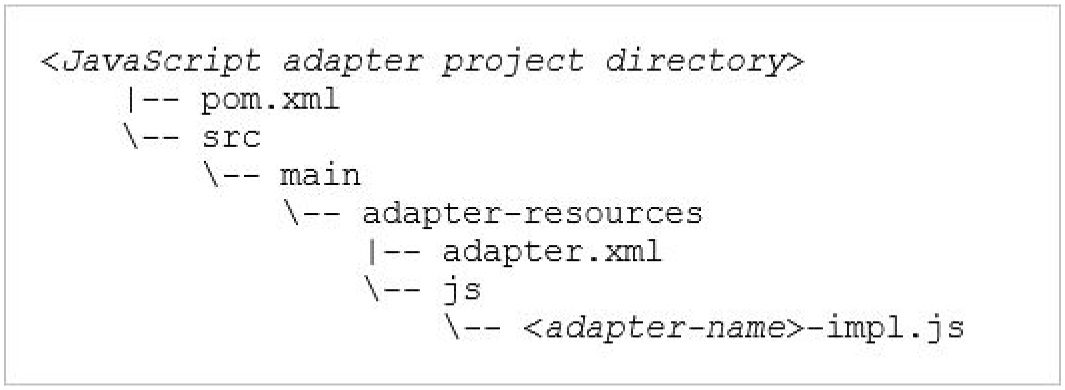
```
<Java adapter project directory>
    |-- pom.xml
    \-- src
        \-- main
            |-- adapter-resources
            |   \-- adapter.xml
            \-- java
                \-- <package>
                    |-- <adapter-name>Application.java
                    \-- <adapter-name>Resource.java
```

Figure 7-8. Java adapter Maven project

- `pom.xml`: The Maven Project Object Model (POM) file is in the root folder. It contains references to the adapter-maven-api, adapter-maven-plugin, and mfp-security-checks-base adapter artifacts that were developed by MobileFirst. For more details, see Maven adapter artifacts.
- `adapter.xml`: The adapter-descriptor file is under `src/main/adapter-resources`. For more details, see "The Java adapter-descriptor file" on page 7-194.
- <*package*>: Java package that contains the JAX-RS application and all the source files that it needs. They are located under `src/main/java/`. For more details, see "Implementing the JAX-RS service of the adapter" on page 7-200.

JavaScript adapter Maven projects

When you create a new JavaScript adapter, a Maven project with the following structure is produced:



```
<JavaScript adapter project directory>
    |-- pom.xml
    \-- src
        \-- main
            \-- adapter-resources
                |-- adapter.xml
                \-- js
                    \-- <adapter-name>-impl.js
```

Figure 7-9. JavaScript adapter Maven project

- `pom.xml`: The Maven Project Object Model (POM) file is in the root folder. It contains references to the adapter-maven-api, adapter-maven-plugin, and mfp-security-checks-base adapter artifacts that were developed by MobileFirst. For more details, see Maven adapter artifacts.
- `adapter.xml`: The adapter-descriptor file is under `src/main/adapter-resources`. For more details, see "The JavaScript adapter-descriptor file" on page 7-206.
- <*adapter-name*>`-impl.js`: Contains the JavaScript source files.

Maven adapter artifacts

MobileFirst adapters contain three main artifacts, which are referenced in the `pom.xml` file:

Maven adapter API

adapter-maven-api contains the MobileFirst code and third-party dependencies that the adapter needs to compile properly.

Maven adapter plug-in

adapter-maven-plugin is a Maven plugin that is used to build the adapter and deploy it. It is also used to pull and push adapter configuration. The plug-in is defined in the `pom.xml` as follows:

```
<plugin>
    <groupId>com.ibm.mfp</groupId>
    <artifactId>adapter-maven-plugin</artifactId>
    <version>8.0.0</version>
    <extensions>true</extensions>
</plugin>
```

The plug-in has four goals: **`build`**, **`deploy`**, **`configpull`**, and **`configpush`**:

- The **`build`** goal creates the adapter file by zipping the relevant files from the project and adding the dependencies to the adapter file. The goal creates the adapter file in the target directory with the name `<`*`adapter-name`*`>.adapter`. The **`build`** goal runs automatically as part of the packaging phase of the Maven lifecycle.
- The **`deploy`** goal deploys the adapter file to the server. To use this goal, ensure that the following values in your `pom.xml` file are correct.

  ```
  <properties>
      <!-- MobileFirst adapter deployment properties -->
      <mfpfUrl>http://localhost:9080/mfpadmin</mfpfUrl>
      <mfpfUser>admin</mfpfUser>
      <mfpfPassword>admin</mfpfPassword>
      <mfpfRuntime>mfp</mfpfRuntime>
  </properties>
  ```

  The parameter `mfpfUrl` is the URL to the server where the adapter is deployed. If you are using HTTPS as the protocol, the connection to the server is encrypted. The parameters `mfpfUser` and `mfpfPassword` are the user name and password of the administration service. The parameter `mfpfRuntime` defines the target runtime to deploy to.

  Note: The adapter-maven-plugin **`deploy`** goal does not support server certificate checking or host name checking. Use it in development only. For production purposes, use the Ant deployer instead. For more information, see "Administering MobileFirst applications through Ant" on page 10-23.

  For more information about deploying adapters with Maven, see Deploying JavaScript adapters and Deploying Java adapters.
- The **`configpull`** goal pulls the configuration for an adapter from the MobileFirst Server and writes it to the file specified for the `mfpfConfigFile` parameter. For more information about configuring adapters in MobileFirst Server, see "Configuring adapters" on page 7-227. To run this goal, do either of the following:

  - Set the `mfpfConfigFile` parameter in the `pom.xml` file to point to an output file name.
  - Pass an output file name by using the **`DmfpfConfigFile`** option in a command. For example:

    ```
    mvn adapter:configpull [-DmfpfConfigFile =<path to file>]
    ```

  The result is a JSON object.
- The **`configpush`** goal pushes the adapter configuration from the file name defined for the `mfpfConfigFile` parameter to the MobileFirst Server. For more information about configuring adapters in MobileFirst Server, see "Configuring adapters" on page 7-227. To run this goal, do either of the following:
  - Set the `mfpfConfigFile` parameter in the `pom.xml` file to point to an output file name.
  - Pass an output file name by using the **`DmfpfConfigFile`** option in a command. For example:

    ```
    mvn adapter:configpush [-DmfpfConfigFile=<path to file>]
    ```

  The result is a JSON object.

Custom security base classes

The mfp-security-checks-base dependency provides base classes that enable you to implement custom security checks in adapters. If your adapter does not use custom security checks, you can delete or comment out this dependency:

```
<dependency>
    <groupId>com.ibm.mfp</groupId>
    <artifactId>mfp-security-checks-base</artifactId>
    <version>8.0.0</version>
</dependency>
```

For more information about implementing custom security, see "Security-checks implementation" on page 7-289.

Third-party Maven dependencies

Use Maven's dependency mechanism to package any artifact into your adapter. For information on limitations in using dependencies, see "Adapters and third-party dependencies" on page 3-27.

Note:

- If the artifact is not a Maven artifact, you can add it as a dependency by using the `systemPath` element. For more information, see Introduction to the Dependency Mechanism.
- If you wish to use the dependency without packaging it in the adapter, use the `provided` scope.

MobileFirst Java adapters

With IBM MobileFirst Platform Foundation, you can create, test, and deploy adapters that are written in Java.

For more general information about MobileFirst adapters, see "Adapters overview" on page 7-187.

Benefits of MobileFirst Java adapters

In addition to benefits that adapters provide in general, Java adapters also provide these:

- Java adapters are based on the JAX-RS specification.
- Java adapters expose a full REST API to the client, giving you full control over the URL structure, the content types, the request and response headers, content, and encoding.
- Java adapters let you define custom MobileFirst security checks for granting client access to protected resources.

The Java adapter framework

Figure 1 illustrates how a mobile device 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. Each of the resources of the adapter has its own URL, so it is possible to protect MobileFirst endpoints using any firewall. 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. 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.

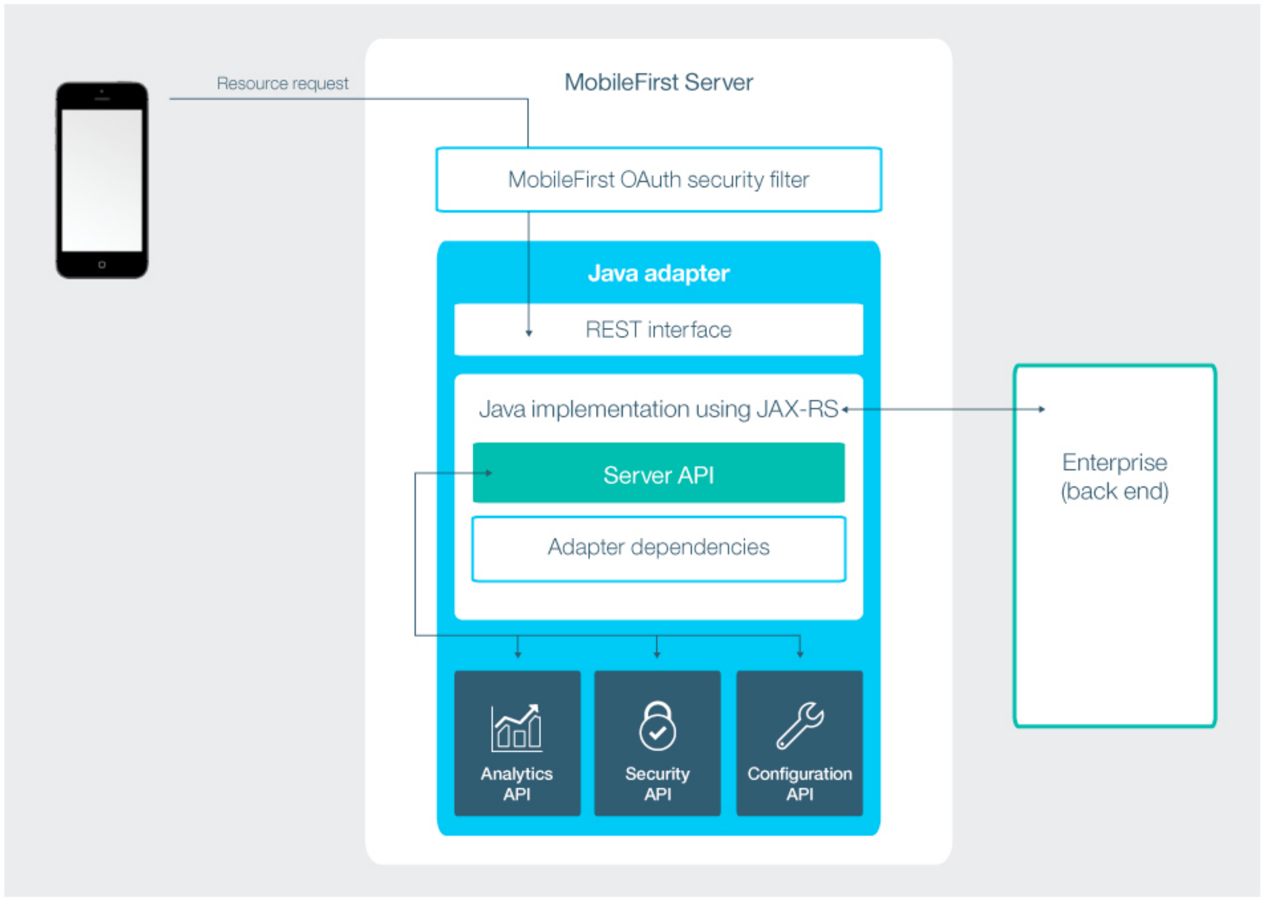


Figure 7-10. The Java adapter framework

The Java adapter-descriptor file

Learn about the function and structure of the Java adapter-descriptor file.

You use the `adapter.xml` descriptor file to declare the display name, description, class name of the JAX-RS application and security checks procedures that are exposed by a Java adapter to applications and to other adapters. The elements, subelements, and attributes of the JavaScript adapter XML file are described in the following sections.

The `adapter.xml` descriptor file is located in the `adapter-resources` folder, under `<`*`Java_adapter`*`>/src/main/`.

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:

```
<?xml version="1.0" encoding="UTF-8"?>
<mfp:adapter name="JavaAdapter1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration">

    <displayName>JavaAdapter1</displayName>
    <description>JavaAdapter1</description>

    <JAXRSApplicationClass>com.acme.JavaAdapter1Application</JAXRSApplicationClass>

    <securityCheckDefinition
        name="UserAuthenticationSC"
        class="com.my_company.package.MyUserAuthenticationSecurityCheck">
        <property
            name="maxAttempts"
            displayName="Maximum attempts"
            defaultValue="3"
            description="Maximum allowed user-authentication attempts"/>
    </securityCheckDefinition>

    <property name="path" description="The path after the host name" defaultValue="/feed">
</mfp:adapter>
```

The `adapter` element 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.

The `adapter` element subelements

The `adapter` element has the following subelements.

`displayName`

Optional.

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.

`JAXRSApplicationClass`

Mandatory for exposing an `/adapter` endpoint.Defines the class name of the JAX-RS application of this adapter. In the example, it is com.acme.JavaAdapter1Application. For more information, see "Implementing the JAX-RS service of the adapter" on page 7-200.

`securityCheckDefinition`

Optional.

Defines a security-check object. For a full reference of the `securityCheckDefinition` element, see "The <securityCheckDefinition> element" on page 7-295.

`property`

Optional.

Declares a user-defined property.

- User-defined properties are shown in the MobileFirst Operations Console in the Configurations tab for the relevant adapter. The values that developers assign to them during the creation of the adapter can be overridden in the console, without redeploying the adapter.
- User-defined properties can be read using the interface and then further customized at run time. For more information, see "Configuring adapters" on page 7-227.

The `property` element has the following structure:

```
<property name="unique-name"
    description="value"
    defaultValue="value"
    type="value"
/>
```

The `property` element has the following attributes.

`name`

Mandatory.

The name of the property. This name must be unique within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter.

`description`

Optional.

A user-friendly description of the property. This is the name that is displayed in the MobileFirst Operations Console.

`defaultValue`

Mandatory.

Defines the default value of this property. This is the value the property will have if it is not overridden.

type

Optional.

The property type. If not specified, a string is assumed. The following values are valid:

- `string`: Default. Any string value is allowed.
- `integer`: Any integer value is allowed.
- `boolean`: Only `true` or `false` are allowed.
- `enumerator`: Only specific values are allowed. This is specified by using a JSON array notation. For example the enumerator `['first', 'second']` allows only the values `first` and `second`.

Working with Java adapters

Learn how to develop and deploy JavaScript adapters.

Creating Java adapters:

You create adapters either with Maven or with the MobileFirst Platform CLI.

Creating adapters with Maven:
Before you begin

To work with Maven, ensure that you have it installed. For more information, see https://maven.apache.org/install.html.

Procedure

In the command line, **cd** to the location for the new Maven project, then type in the following command:

```
mvn archetype:generate
-DarchetypeGroupId=com.ibm.mfp
-DarchetypeArtifactId=adapter-maven-archetype-java
-DarchetypeVersion=<latest_version>
-DgroupId=<created_project_groupId>
-DartifactId=<created_project_artifactId>
-Dpackage=<created_project_java_package>
-Dversion=<created_project_version>
```

The following is an explanation of the parameters for the **archetype:generate** command:

archetypeGroupId
Archetype group ID. Identifies the MobileFirst adapter archetype template. Specify **com.ibm.mfp** in all cases.

archetypeArtifactId
Archetype artifact ID. Identifies the MobileFirst adapter archetype template. Specify **adapter-maven-archetype-java**.

archetypeVersion
Archetype version. Identifies the MobileFirst adapter archetype template. Specify the latest version that is available in the repository.

groupId
Sets the group of the new Maven project. Specify your own value. For more information, see What is the POM?.

artifactId
Sets the artifact ID of the new Maven project. This value will later be used as the adapter name. Specify your own value.

Note: The artifact ID can contain alphanumeric characters and underscores, and must start with a letter.

package
Sets the Java package name in `src/main/java`. Specify your own value. The default is <*groupId*>.

version
Sets the version of the new Maven project. Set your own value. The default is **`1.0-SNAPSHOT`**.

Creating adapters with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure

In the command prompt, run the following command:

```
mfpdev adapter create <adapter_name> -t <adapter_type> -p <adapter_package_name> -g <maven_project_groupid>
```

The following is an explanation of the parameters for the **`mfpdev adapter create`** command:

adapter_name
Adapter name. Specify a value.

adapter_type
Adapter type. Specify **`Java`**.

adapter_package_name
Java adapter package name in `src/main/java`. Specify a value.

maven_project_groupid
Group ID of the Maven project.

Creating adapters with the MobileFirst Operations Console:
Procedure

You can also create an adapter in the MobileFirst Operations Console by clicking **New** in the **Adapters** area and following the instructions.

Building Java adapters:

You build Java adapters with Maven or MobileFirst Platform CLI.

Building adapters with Maven:
Before you begin

To work with Maven, ensure that you have it installed. For more information, see https://maven.apache.org/install.html.

Procedure

At the command prompt, run the **`mvn install`** command to build the Maven project. An `.adapter` archive file is generated in the `target` folder.

Building adapters with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure

1. At the command prompt, **`cd`** to the root folder of the Maven project.
2. Run **`mfpdev adapter build`** to build the Maven project. An `.adapter` archive file is generated in the `target` folder.

   Tip: You can also find and build all of the adapters that are in the current directory and its subdirectories by entering **`mfpdev adapter build all`** while you are in that directory.

Deploying Java adapters:

You deploy a Java adapter with Maven, or the MobileFirst Platform CLI.

Configuring the **`deploy`** *goal:*
About this task

The **`deploy`** goal deploys the adapter file to the server. Before you deploy the adapter, ensure that the values in your `pom.xml` file that relate to the **`deploy`** goal are correct. For more information, see The deploy goal.

Deploying adapters with Maven:
Before you begin

To work with Maven, ensure that you have it installed. For more information, see https://maven.apache.org/install.html.

Procedure

1. At the command prompt, navigate to the root folder of the project.
2. Run one of the following commands:
   - **`mvn adapter:deploy`** to deploy the adapter.
   - **`mvn install adapter:deploy`** to build and deploy the adapter.

   Note: The **`deploy`** command is available only during development.

Deploying adapters with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure

1. At the command prompt, navigate to the root folder of the project.
2. Run the **`mfpdev adapter deploy -x`** command to deploy the adapter.

The -x option deploys the adapter to the MobileFirst Server that is specified in the `pom.xml` file of the adapter. If you leave out the option, CLI uses the default server that is specified in the CLI settings.

Tip: You can also find and deploy all of the adapters that are in the current directory and its subdirectories by entering **`mfpdev adapter deploy all`** while you are in that directory.

For more CLI deployment options, run the command: **`mfpdev help adapter deploy`**.

Note: The **`deploy`** command is available only during development.

Deploying adapters with the MobileFirst Operations Console:
Procedure

You can also deploy an adapter in the MobileFirst Operations Console by selecting **Deploy Adapter** in the **Actions** menu and following the instructions.

Pushing Java adapter configurations:

You push Java adapter configurations to the server with Maven or MobileFirst Platform CLI.

Pushing adapter configurations with Maven:
Before you begin

To work with Maven, ensure that you have it installed and that the **`mvn`** command is identified in your system path. For more information about installing Maven, see https://maven.apache.org/install.html.

Procedure

See "Maven adapter artifacts" on page 7-191 for the procedure for pushing the configurations with Maven.

Pushing adapter configurations with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure
1. At the command prompt, **`cd`** to the root directory of the adapter project. This directory was created with your adapter, and has the same name as the adapter. The `config.json` file is in this directory.
2. Run **`mfpdev adapter push`** to push the adapter configuration to the default server.

Pulling Java adapter configurations:

You pull Java adapter configurations from the server with Maven or MobileFirst Platform CLI.

Pulling adapter configurations with Maven:

Before you begin

To work with Maven, ensure that you have it installed and that the **`mvn`** command is identified in your system path. For more information about installing Maven, see https://maven.apache.org/install.html.

Procedure

See "Maven adapter artifacts" on page 7-191 for the procedure for pulling the configurations with Maven.

Pulling adapter configurations with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure
1. At the command prompt, **`cd`** to the root directory of the adapter project. This directory was created with your adapter, and has the same name as the adapter. If the `config.json` file is not in this directory, it is created.
2. Run **`mfpdev adapter pull`** to pull the adapter configuration from the default server.

Developing Java adapter code

Learn about implementing the JAX-RS service in the adapter and the Java server side API.

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.

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;
    }
}
```

Note: The example demonstrates how to write a pure JAX-RS application using getClasses and getSingletons. A quicker alternative is to use **`extends MFPJAXRSApplication`**. The MFPJAXRSApplication class scans the package for JAX-RS 2.0 resources and automatically creates a list. Additionally, its init method is called by MobileFirst Server as soon as the adapter is deployed, before it starts serving, and when the MobileFirst runtime starts up.

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 7-18. Resource requests.

Request	Description
GET /users	Gets all users list
POST /user	Adds a new user
GET /users/{userId}	Gets a specific user with id **userId**
PUT /users	Updates an existing user
DELETE /users/{userId}	Deletes a user with id **userId**

The JAX-RS framework does the mapping from the simple Java object User to a JSON object and conversely, thereby 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.

Configuring protection of JAX-RS resources:
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 access token. See "OAuth resource protection" on page 7-271. You can configure the resource protection by using the @OAuthSecurity annotation to assign a custom security scope, or to disable resource protection. For detailed configuration instructions, see Configure protection of Java API for RESTful Web Services (JAX-RS) resources.

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, getting values of configuration properties, reporting activities to IBM MobileFirst Analytics, and getting the identity of the request issuer.

The Java API has four interfaces, corresponding to four categories: authentication, adapters, configuration, and analytics:

- AdapterSecurityContext
- AdaptersAPI
- ConfigurationAPI
- AnalyticsAPI

To access an interface, use the `@` annotation. For example, to access the AdaptersAPI interface, write:

```
@Context
AdaptersAPI adaptersApi;
```

Examples of the use of the API are provided in the following sections.

AdapterSecurityContext

The security API provides access to the security context of the client, and the client registration data. To use the API, add an instance of AdapterSecurityContext with the `@Context` annotation. The following sample uses the API to get the display name of the authenticated user:

```
@Context
AdapterSecurityContext securityContext;

@OAuthSecurity(scope = "userLogin")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String sayHello() {
   AuthenticatedUser user = securityContext.getAuthenticatedUser();
   return "Hello " + user.getDisplayName();
}
```

AdaptersAPI

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 access to AdaptersAPI
    @Context AdaptersAPI adaptersAPI;
    @GET
    @Path("/calljs")
    @Produces("application/json")
    public JSONObject callJSAdapterExample() throws Exception {
        //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
        return json;
    }
```

ConfigurationAPI

The configuration API enables the adapter to read server-side configuration properties. These properties are defined in one of two places: as adapter configuration or as JNDI entries.

For example, assume you have a user-defined property, databaseName. To get its value, you could write the following code:

```
String databaseName = configurationApi.getPropertyValue("databaseName");
```

For more information, see "Configuring adapters" on page 7-227.

AnalyticsAPI

The Analytics API reports information to IBM MobileFirst Analytics.

For example, to send the string `Getting account balance`, you might write:

```
@Context
AnalyticsAPI analyticsApi;
@GET
public String customScopeProtected() {
    analyticsApi.logActivity("Getting account balance");
    // perform operation
    }
```

Implementing connectivity in Java adapters

Unlike JavaScript adapters, Java adapters are not provided with built-in connectivity. You can implement connectivity by using custom properties in the adapter descriptor file. For more information, see "Configuring adapters" on page 7-227. The Java Adapters tutorial on the Developer Center website demonstrates this feature.

MobileFirst JavaScript adapters

With IBM MobileFirst Platform Foundation, you can create, test, and deploy adapters that are written in JavaScript.

For more general information about MobileFirst adapters, see "Adapters overview" on page 7-187.

Benefits of MobileFirst JavaScript adapters

In addition to benefits that adapters provide in general, JavaScript adapters also provide these:

- **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.
- **REST Interface**: With the REST interface, you can benefit from the OAuth 2.0 security framework. For more information, see "Client access to adapters" on page 7-231.

JavaScript adapter types

IBM MobileFirst Platform Foundation V8.0.0 supports HTTP and SQL adapters:

- With a HTTP adapter, you can send GET or POST HTTP requests and retrieve data from the response headers and body. HTTP adapters work with RESTful and SOAP-based services, and can read structured HTTP sources such as RSS feeds.

- With an SQL adapter, you can communicate with any SQL data source. You can use plain SQL queries or stored procedures.

The JavaScript adapter framework

The adapter framework mediates between the mobile apps and the enterprise. A typical flow is depicted in the following diagram.

- The connectivity and auto-conversions are facilities that are provided with IBM MobileFirst Platform Foundation.
- The back-end application, JavaScript code and XSLT components in the MobileFirst Server are supplied by the adapter or app developer.

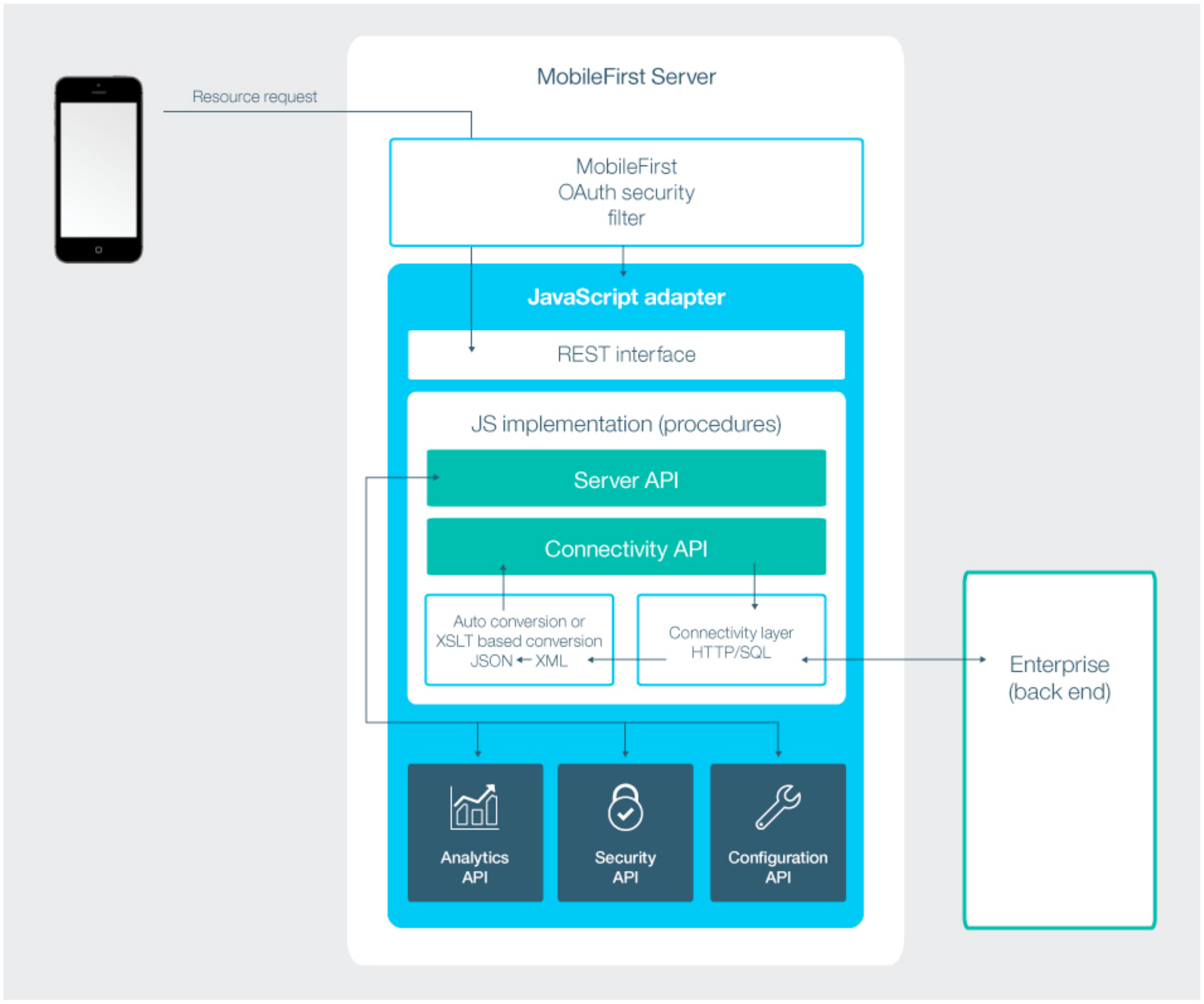


Figure 7-11. The JavaScript adapter framework

- An adapter provides a set of services, called procedures. Mobile apps invoke procedures by issuing Ajax requests.
- The procedure retrieves information from the back-end application.
- 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 (XSLT) 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.

- The JavaScript implementation of the procedure receives the JSON data, performs any additional processing, and returns it to the calling app.

Note:

- Writing an adapter that pulls large amounts of data and transfers it to the client application is discouraged because the data must be processed twice: once at the adapter and once again at the client application.
- 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 first be encoded in Base64 format.

The JavaScript adapter-descriptor file

Learn about the function and structure of the Java adapter-descriptor file.

You use the `adapter.xml` descriptor file to configure adapter connectivity to the back-end system and to declare the procedures that are exposed by the adapter to applications and to other adapters. The elements, subelements, and attributes of the JavaScript adapter XML file are described in the following sections.

The `adapter.xml` descriptor file is located in the `adapter-resources` folder, under `<`*`JavaScript_adapter`*`>/src/main/`.

The `adapter` element

The `adapter` element is the root element of the adapter configuration file. The following example shows the structure, which consists of both attributes and subelements:

```
<mfp:adapter name="Backend_adap"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:mfp="http://www.ibm.com/mfp/integration"
  xmlns:http="http://www.ibm.com/mfp/integration/http">

  <displayName>Backend_adap</displayName>
  <description>Backend_adap</description>
  <connectivity>
    <connectionPolicy>...</connectionPolicy>
  </connectivity>
</mfp:adapter>
```

The `adapter` element attributes

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.

The `adapter` element subelements

The `adapter` element has the following subelements.

`displayName`

Optional.

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. For more information about `connectionPolicy`, see “HTTP adapter connectionPolicy element” on page 7-209 and “SQL adapter connectionPolicy element” on page 7-213.

`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. An example is shown in Implementing JavaScript SQL adapters.

The `procedure` element has the following structure:

```
<procedure
    name="unique-name"
    audit="value"
    scope="value"
    secured="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.

`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.

`scope`

Optional.

The security scope that protects the adapter resource procedure, as a string of zero or more space-separated scope elements. A scope element can be a keyword that is mapped to a security check, or the name of a security check. The default value of the `scope` attribute is an empty string. When the value of the `secured` attribute is `false`, the `scope` attribute is ignored.

For information on OAuth resource protection, see "OAuth resource protection" on page 7-271. For detailed instructions on how to configure resource protection for a JavaScript resource procedure, including how to configure a protecting scope, see Configure protection of JavaScript resources.

`secured`

Optional.

Defines whether the adapter resource procedure is protected by the MobileFirst security framework. The following values are valid:

- `true`: Default. The procedure is protected. Calls to the procedure require a valid access token.
- `false`: The procedure is not protected. Calls to the procedure do not require an access token. When this value is set, the `scope` attribute is ignored. To understand the implications of disabling resource protection, see "Unprotected resources" on page 7-271.

For information on OAuth resource protection, see "OAuth resource protection" on page 7-271. For detailed instructions on how to configure resource protection for a JavaScript resource procedure, including how to disable resource protection, see Configure protection of JavaScript resources.

`securityCheckDefinition`

Optional.

Defines a security-check object. For a full reference of the `securityCheckDefinition` element, see "The <securityCheckDefinition> element" on page 7-295.

`property`

Optional.

Declares a user-defined property.

- User-defined properties are shown in the MobileFirst Operations Console in the Configurations tab for the relevant adapter. The values that developers assign to them during the creation of the adapter can be overridden in the console, without redeploying the adapter.
- User-defined properties can be read using the and then further customized at run time. For more information, see "Configuring adapters" on page 7-227.

The `property` element has the following structure:

```
<property name="unique-name"
    description="value"
    defaultValue="value"
    type="value"
/>
```

Note: If the `adapter.xml` file of a JavaScript adapter contains `<procedure>` elements, the `<property>` elements must always appear **below** them.
The `property` element has the following attributes.

`name`

Mandatory.

The name of the property. This name must be unique within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter.

`description`

Optional.

A user-friendly description of the property. This is the name that is displayed in the MobileFirst Operations Console.

`defaultValue`

Mandatory.

Defines the default value of this property. This is the value the property will have if it is not overridden.

`type`

Optional.

The property type. If not specified, a string is assumed. The following values are valid:

- `string`: Default. Any string value is allowed.
- `integer`: Any integer value is allowed.
- `boolean`: Only `true` or `false` are allowed.
- `enumerator`: Only specific values are allowed. This is specified by using a JSON array notation. For example the enumerator `['first', 'second']` allows only the values `first` and `second`.

HTTP adapter connectionPolicy element:

The `connectionPolicy` element of the adapter-descriptor file lets you configure settings for your adapter's HTTP connection.

This page describes only the `connectionPolicy` element of the adapter-descriptor file. For information about other elements, see "The JavaScript adapter-descriptor file" on page 7-206.

Structure

The following example of a `connectionPolicy` element shows its structure.

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType"; cookiePolicy="cookie-policy" maxRedirects="10">
    <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 `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.

- `BEST_MATCH`: default value
- `BROWSER_COMPATIBILITY`
- `RFC_2109`
- `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 "Using SSL in HTTP adapters" on page 7-222.

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 "Using SSL in HTTP adapters" on page 7-222.

authentication

Optional.

Authentication configuration of the HTTP adapter. The HTTP adapter can use one of two authentication protocols. Define the `<authentication>` element, as follows:

- Basic authentication

  ```
  <authentication>
    <basic/>
  </authentication>
  ```
- Digest authentication

  ```
  <authentication>
    <digest/>
  </authentication>
  ```

The connection policy can contain a `<serverIdentity>` element. This feature applies to all authentication schemes. For example:

```
<authentication>
    <basic/>
    <serverIdentity>
        <username></username>
        <password></password>
    </serverIdentity>
</authentication>
```

proxy

Optional.

The `proxy` element specifies the details of the proxy server to use when accessing the back-end application. Add a `proxy` element inside the `connectionPolicy` element. The proxy details must include the protocol domain and port. 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. For more information, see the 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 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 its accompanying 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.

SQL adapter connectionPolicy element:

The `connectionPolicy` element of the adapter-descriptor file lets you configure settings for your adapter's SQL connection.

This page describes only the `connectionPolicy` element of the adapter-descriptor file. For information about other elements, see “The JavaScript adapter-descriptor file” on page 7-206.

Structure

The `connectionPolicy` element has two options for connection:
- The `dataSourceDefinition` subelement for development mode
- The `dataSourceJNDIName` subelement for production mode

Attributes

The `connectionPolicy` element has one attribute:

`xsi:type`

Mandatory.

The value of this attribute must be set to `sql:SQLConnectionPolicy`.

Subelements

The `connectionPolicy` element must contain one of the following two subelements.

`dataSourceDefinition`

Optional

Contains the parameters that are needed to connect to a data source. The adapter creates a connection for each request.

For example:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
    <dataSourceDefinition>
        <driverClass>com.mysql.jdbc.Driver</driverClass>
        <url>jdbc:mysql://localhost:3306/mysqldbname</url>
        <user>user_name</user>
        <password>password</password>
    </dataSourceDefinition>
</connectionPolicy>
```

dataSourceJNDIName

Optional.

Connect to the data source by using the JNDI name of a data source that is provided by the application server. The adapter takes the connection from the server connection pool that is associated with the JNDI name.

Application servers provide a way to configure data sources. For more information, see "Installing MobileFirst Server to an application server" on page 6-100. For example:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceJNDIName>my-adapter-ds</dataSourceJNDIName>
</connectionPolicy>
```

Working with JavaScript adapters

Learn how to develop, build, deploy, push, and pull Java adapters.

Creating JavaScript adapters:

You create adapters with Maven, the MobileFirst Platform CLI, or the MobileFirst Operations Console.

Creating adapters by using Maven:
Before you begin

To work with Maven, ensure that you have it installed. For more information, see https://maven.apache.org/install.html.

Procedure

In the command line, **cd** to the location for the new Maven project, then type in the following command:

```
mvn archetype:generate
-DarchetypeGroupId=com.ibm.mfp
-DarchetypeArtifactId=<adapter-maven-archetype-type>
-DarchetypeVersion=<latest_version>
-DgroupId=<created_project_groupId>
-DartifactId=<created_project_artifactId>
-Dversion=<created_project_version>
```

The following is an explanation of the parameters for the **archetype:generate** command:

archetypeGroupId
Archetype group ID. Identifies the MobileFirst adapter archetype template. Specify **com.ibm.mfp** in all cases.

archetypeArtifactId
Archetype artifact ID. Identifies the MobileFirst adapter archetype template. Specify one of the following:

- **adapter-maven-archetype-http** to create a JavaScript adapter with HTTP connectivity
- **adapter-maven-archetype-sql** to create a JavaScript adapter with SQL connectivity

archetypeVersion
Archetype version. Identifies the MobileFirst adapter archetype template. Specify the latest version available in the repository.

groupId
Sets the group of the new Maven project. Specify your own value. For more information, see What is the POM?

artifactId
Sets the artifact ID of the new Maven project. This value will later be used as the adapter name. Specify your own value.

Note: The artifact ID can contain alphanumeric characters and underscores, and must start with a letter.

version
Sets the version of the new Maven project. Set your own value. The default is **`1.0-SNAPSHOT`**.

Creating adapters using MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure

In the command prompt, run the following command:

```
mfpdev adapter create <adapter_name> -t <adapter_type>
-p <adapter_package_name> -g <maven_project_groupid>
```

The following is an explanation of the parameters for the **`mfpdev adapter create`** command:

adapter_name
Adapter name. Specify a value.

adapter_type
Adapter type. Specify one of the following:
- **`HTTP`** to create a JavaScript adapter with HTTP connectivity
- **`SQL`** to create a JavaScript adapter with SQL connectivity

maven_project_groupid
Group ID of the Maven project.

Creating adapters with the MobileFirst Operations Console:
Procedure

You can also create an adapter in the MobileFirst Operations Console by clicking **New** in the **Adapters** area and following the instructions.

Building JavaScript adapters:

You build JavaScript adapters with Maven or MobileFirst Platform CLI.

Building adapters with Maven:
Before you begin

To work with Maven, ensure that you have it installed. For more information, see https://maven.apache.org/install.html.

Procedure

At the command prompt, run the **mvn install** command to build the Maven project. An `.adapter` archive file is generated in the `target` folder.

Building adapters with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure

1. At the command prompt, **cd** to the root folder of the Maven project.
2. Run **mfpdev adapter build** to build the Maven project. An `.adapter` archive file is generated in the `target` folder.

   Tip: You can also find and build all of the adapters that are in the current directory and its subdirectories by entering **mfpdev adapter build all** while you are in that directory.

Deploying JavaScript adapters:

You deploy a JavaScript adapter with Maven, or the MobileFirst Platform CLI.

Configuring the **deploy** *goal:*
About this task

The **deploy** goal deploys the adapter file to the server. Before you deploy the adapter, ensure that the values in your `pom.xml` file that relate to the **deploy** goal are correct. For more information, see The deploy goal.

Deploying adapters with Maven:
Before you begin

To work with Maven, ensure that you have it installed. For more information, see https://maven.apache.org/install.html.

Procedure

1. At the command prompt, navigate to the root folder of the project.
2. Run one of the following commands:
   - **mvn adapter:deploy** to deploy the adapter.
   - **mvn install adapter:deploy** to build and deploy the adapter.

   Note: The **deploy** command is available only during development.

Deploying adapters with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure

1. At the command prompt, navigate to the root folder of the project.
2. Run the **mfpdev adapter deploy -x** command to deploy the adapter.

The -x option deploys the adapter to the MobileFirst Server that is specified in the `pom.xml` file of the adapter. If you leave out the option, CLI uses the default server that is specified in the CLI settings.

Tip: You can also find and deploy all of the adapters that are in the current directory and its subdirectories by entering **`mfpdev adapter deploy all`** while you are in that directory.

For more CLI deployment options, run the command: **`mfpdev help adapter deploy`**.

Note: The **`deploy`** command is available only during development.

Deploying adapters with the MobileFirst Operations Console:
Procedure

You can also deploy an adapter in the MobileFirst Operations Console by selecting **Deploy Adapter** in the **Actions** menu and following the instructions.

Pushing JavaScript adapter configurations:

You push JavaScript adapter configurations to the server with Maven or MobileFirst Platform CLI.

Pushing adapter configurations with Maven:
Before you begin

To work with Maven, ensure that you have it installed and that the **`mvn`** command is identified in your system path. For more information about installing Maven, see https://maven.apache.org/install.html.

Procedure

See "Maven adapter artifacts" on page 7-191 for the procedure for pushing the configurations with Maven.

Pushing adapter configurations with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure
1. At the command prompt, **`cd`** to the root directory of the adapter project. This directory was created with your adapter, and has the same name as the adapter. The `config.json` file is in this directory.
2. Run **`mfpdev adapter push`** to push the adapter configuration to the default server.

Pulling JavaScript adapter configurations:

You pull JavaScript adapter configurations from the server with Maven or MobileFirst Platform CLI.

Pulling adapter configurations with Maven:

Before you begin

To work with Maven, ensure that you have it installed and that the **`mvn`** command is identified in your system path. For more information about installing Maven, see https://maven.apache.org/install.html.

Procedure

See "Maven adapter artifacts" on page 7-191 for the procedure for pulling the configurations with Maven.

Pulling adapter configurations with MobileFirst Platform CLI:
Before you begin

Ensure that you have installed MobileFirst Platform CLI and Maven.

Procedure
1. At the command prompt, **`cd`** to the root directory of the adapter project. This directory was created with your adapter, and has the same name as the adapter. If the `config.json` file is not in this directory, it is created.
2. Run **`mfpdev adapter pull`** to pull the adapter configuration from the default server.

Developing JavaScript adapter code

Learn about implementing JavaScript adapters.

JavaScript adapters and global variables

The IBM MobileFirst Platform Server does not rely on HTTP sessions. In a deployment that involves multiple nodes, a client request can be processed by one server in a cluster and the next request by another. You should not rely on global variables to keep data from one request to the next.

Adapter response threshold

Adapter calls are not designed to return huge chunks of data because the adapter response is stored in MobileFirst Server memory as a string. Thus, data that exceeds the amount of available memory might cause an out-of-memory exception and the failure of the adapter invocation. To prevent such failure, you configure a threshold value from which the MobileFirst Server returns gzipped HTTP responses. The HTTP protocol has standard headers to support gzip compression. The client application must also be able to support gzip content in HTTP.

Server side

In the MobileFirst Operations Console, under **Runtimes** > **Settings** > **GZIP compression threshold for adapter responses**, set the desired threshold value and save. The default value is 20 KB.

Note: By saving the change in the MobileFirst Operations Console, the change is effective immediately in the runtime.

Client side

Ensure that you enable the client to parse a gzip response, by setting the value of the `Accept-Encoding` headers to `gzip` in every client request. Examples follow:

Android native apps

```
WLResourceRequest req = new WLResourceRequest(new URI("/adapters/sampleAdapter/procedure"), WLResourceRequest.GET);
req.addHeader("Accept-Encoding", "gzip");
req.send(myListener);
```

Javascript for Cordova apps

```
var request = new WLResourceRequest('/adapters/sampleAdapter/procedure', WLResourceRequest.GET);
 request.addHeader('Accept-Encoding','gzip');
 request.send().then(
         function(response) {
             // success flow, the result can be found in response.responseJSON
         },
         function(error) {
             // failure flow
             // the error code and description can be found in error.errorCode and error.errorMsg fields respectively
         }
 );
```

Implementing JavaScript HTTP adapters:

Learn to develop a JavaScript HTTP adapter.

Before you begin

The example here shows how to implement an adapter that connects with back-end HTTP services by using the connectivity facilities that are provided with MobileFirst Server. You can learn more about connectivity in “The JavaScript adapter framework” on page 7-205 and “HTTP adapter connectionPolicy element” on page 7-209.

HTTP adapters work with RESTful and SOAP-based services, and can read structured HTTP sources such as RSS feeds.

You can easily customize HTTP adapters with simple server-side JavaScript code. For example, you can set up server-side filtering if necessary. The retrieved data can be in XML, HTML, JSON, or plain text format.

There are three parts to the implementation of a JavaScript adapter:
- Configuring the `adapter.xml` descriptor file:
  - In the `connectionPolicy` element, you declare the parameters that relate to the HTTP connection over which the adapter connects. For more information, see “HTTP adapter connectionPolicy element” on page 7-209.
  - You declare each procedure that you implement in the JavaScript source files, by using a `procedure` element. For more information, see HTTP adapter procedure element.
- Implementing procedure logic in JavaScript source files.
- (Optional) Using XSL to filter received records and fields.

This page also shows you how to call a SOAP-based service in the HTTP adapter.

Configuring the `adapter.xml` descriptor file:
Procedure

1. In the adapter-descriptor file, configure the following parameters inside the `connectivity`element:
   - Set the protocol to `http` or `https`.
   - Set the HTTP domain to the domain part of the HTTP URL.
   - Set the TCP Port.

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <mfp:adapter name="JavaScriptHTTP"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ```

```
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>JavaScriptHTTP</displayName>
    <description>JavaScriptHTTP</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>https</protocol>
            <domain>mobilefirstplatform.ibmcloud.com</domain>
            <port>443</port>
            <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
            <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
            <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
        </connectionPolicy>
    </connectivity>

</mfp:adapter>
```

2. Declare a getFeed procedure to get an RSS feed and a getFeedFiltered procedure to specify XSL transformation options on the feed data.

   Note: The name of the procedure that you declare in the adapter-descriptor file must be identical to the name you use when you implement the procedure itself.

   ```
   <procedure name="getFeed"/>
   <procedure name="getFeedFiltered"/>
   ```

JavaScript procedure implementation:
Before you begin

A service URL is used for procedure invocations. Some parts of the URL are constant. For example, `http://example.com/`. 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 more information about advanced adapter options, such as cookies, headers, and encoding, see "HTTP adapter connectionPolicy element" on page 7-209.

Procedure

Call an HTTP request by using the MFP.Server.invokeHttp function. Note that MFP.Server.invokeHttp requires 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 the MFP.Server.invokeHttp function.

```
function getFeed() {
  var input = {
      method : 'get',
```

```
    returnedContentType : 'xml',
    path : "feed.xml"
  };

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

XSL transformation filtering:
Before you begin

You can also apply XSL transformation to the received data. For example, to filter the feed data.

Procedure

1. Create a `filtered.xsl` file in the same location as the JavaScript implementation file.
2. Specify the transformation options in the input parameters of the getFeedFiltered procedure invocation.

   ```
   function getFeedFiltered() {

     var input = {
         method : 'get',
         returnedContentType : 'xml',
         path : "feed.xml",
         transformation : {
           type : 'xslFile',
           xslFile : 'filtered.xsl'
         }
     };

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

Creating a SOAP-based service request*:*
Before you begin

You can use the MFP.Server.invokeHttp function to create a SOAP envelope.

Procedure

1. To call a SOAP-based service in a JavaScript HTTP adapter, encode the SOAP XML envelope within the request body by using ECMAScript for XML (E4X).

   ```
   var request =
           <soap:Envelope
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
                  <soap:Body>
                      <GetCitiesByCountry xmlns="http://www.webserviceX.NET">
                           <CountryName>{countryName}</CountryName>
                      </GetCitiesByCountry>
                  </soap:Body>
           </soap:Envelope>;
   ```
2. Use the MFP.Server.invokeHttp (options) function to call a request for a SOAP service. The **`options`** argument is a JSON object that must include the following properties:
   - A **`method`** property: usually POST
   - A **`returnedContentType`** property: usually XML
   - A **`path`** property: a service path
   - A **`body`** property: content (SOAP XML as a string) and contentType

```
var input = {
    method: 'post',
    returnedContentType: 'xml',
    path: '/globalweather.asmx',
    body: {
        content: request.toString(),
        contentType: 'text/xml; charset=utf-8'
    }
};

var result = MFP.Server.invokeHttp(input);
```

Using SSL in HTTP adapters:

You can use SSL in an HTTP adapter with simple and mutual authentication to connect to back-end services.

About this task

Configure the MobileFirst Server to use SSL in an HTTP adapter by performing the steps described here.

Note: SSL represents transport level security, which is independent of basic authentication. It is possible to do basic authentication either over HTTP or HTTPS.

Procedure

1. Set the URL protocol of the HTTP adapter to `https`.
2. Store SSL certificates in the MobileFirst Server keystore. See "Configuring the MobileFirst Server keystore" on page 7-316.

SSL with mutual authentication

If you use SSL with mutual authentication, you must also perform the following steps:

3. Generate your own private key for the HTTP adapter or use one provided by a trusted authority.
4. If you generated your own private key, export the public certificate of the generated private key and import it into the back-end truststore.
5. Define an alias and password for the private key in the `connectionPolicy` element of the adapter-descriptor XML file, `adapter.xml`. The `sslCertificateAlias` and `sslCertificatePassword` subelements are described in "HTTP adapter connectionPolicy element" on page 7-209.

Implementing JavaScript SQL adapters:

Learn to develop a JavaScript SQL adapter.

Before you begin

The example in this page shows how to implement an adapter that connects with a MySQL database by using the connectivity facilities that are provided with MobileFirst Server. You can learn more about connectivity in "The JavaScript adapter framework" on page 7-205 and "SQL adapter connectionPolicy element" on page 7-213.

To connect to an SQL database, JavaScript code needs a JDBC connector driver for the specific database type. You must download the appropriate JDBC connector driver and add it as a dependency in your Maven project. For more information about adding dependencies, see System Dependencies.

There are two parts to the implementation of a JavaScript adapter:

- Configuring the `adapter.xml` descriptor file:
  - In the `connectionPolicy` element, you declare the parameters of the SQL database to which the adapter connects. For more information, see "SQL adapter connectionPolicy element" on page 7-213.
  - You declare each procedure that you implement in the JavaScript source files, by using a `procedure` element. For more information, see SQL adapter procedure element.
- Implementing procedure logic in JavaScript source files.

Configuring the `adapter.xml` descriptor file:

Procedure

1. In the adapter-descriptor file, inside the `connectivity`element:, configure the following parameters:
   - JDBC driver class
   - Database URL
   - Username
   - Password

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <mfp:adapter name="JavaScriptSQL"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mfp="http://www.ibm.com/mfp/integration"
       xmlns:sql="http://www.ibm.com/mfp/integration/sql">

       <displayName>JavaScriptSQL</displayName>
       <description>JavaScriptSQL</description>
       <connectivity>
           <connectionPolicy xsi:type="sql:SQLConnectionPolicy">
               <dataSourceDefinition>
                   <driverClass>com.mysql.jdbc.Driver</driverClass>
                   <url>jdbc:mysql://localhost:3306/mobilefirst_training</url>
                   <user>mobilefirst</user>
                   <password>mobilefirst</password>
               </dataSourceDefinition>
           </connectionPolicy>
       </connectivity>

   </mfp:adapter>
   ```
2. Declare a procedure for connecting to the database with a plain SQL query and one for connecting with a stored procedure.

   Note: The name of the procedure that you declare in the adapter-descriptor file must be identical to the name you use when implementing the procedure itself.

   ```
   <procedure name = "getAccountTransactions1" />
   <procedure name = "getAccountTransactions2" />
   ```

JavaScript procedure implementation: SQL statement query:

Procedure

1. Assign your SQL query to a variable.
2. Add parameters, if necessary.

```
var getAccountsTransactionsStatement = "SELECT transactionId, fromAccount, toAccount, transactionDate, transactionAmount
    "FROM accounttransactions " +
    "WHERE accounttransactions.fromAccount = ? OR accounttransactions.toAccount = ? " +
    "ORDER BY transactionDate DESC " +
    "LIMIT 20;";
```

3. Use the MFP.Server.invokeSQLStatement function to call prepared queries.

```
    function getAccountTransactions1(accountId){
        // MFP.Server.invokeSQLStatement calls prepared queries
```

4. Return the result to the application or to another procedure.

```
return MFP.Server.invokeSQLStatement({
    preparedStatement : getAccountsTransactionsStatement,
    parameters : [accountId, accountId]
});
```

JavaScript procedure implementation: SQL stored procedure:
Procedure

Run an SQL stored procedure by using the MFP.Server.invokeSQLStoredProcedure function. Specify an SQL stored procedure name as an invocation parameter.

```
// Invoke stored SQL procedure and return invocation result
function getAccountTransactions2 ( accountId ){
    // To run a SQL stored procedure, use the MFP.Server.invokeSQLStoredProcedure method
    return MFP . Server . invokeSQLStoredProcedure ({
        procedure : "getAccountTransactions" ,
        parameters : [ accountId ]
    });
}
```

Using multiple parameters:
Procedure

When using multiple parameters in an SQL query, make sure to accept the variables in the function and pass them to the MFP.Server.invokeSQLStatement or MFP.Server.invokeSQLStoredProcedure parameters in an array.

```
var getAccountsTransactionsStatement = "SELECT transactionId, fromAccount, toAccount, transactionDate, transactionAmoun
    "FROM accounttransactions " +
    "WHERE accounttransactions.fromAccount = ? AND accounttransactions.toAccount = ? " +
    "ORDER BY transactionDate DESC " +
    "LIMIT 20;";

//Invoke prepared SQL query and return invocation result
function getAccountTransactions1(fromAccount, toAccount){
    return MFP.Server.invokeSQLStatement({
        preparedStatement : getAccountsTransactionsStatement,
        parameters : [fromAccount, toAccount]
    });
}
```

Output: JSON object:
Results

Assuming that the following is the result set:

Table 7-19. Database entries

fromAccount	toAccount	transactionAmount	transactionDate	transactionId	transactionType
"12345"	"54321"	180.00	"2009-03-11T11:08:39.000Z"	"W06091500863"	"Funds Transfer"
"12345"	null	130.00	"2009-03-07T11:09:39.000Z"	"W214122\/5337"	"ATM Withdrawal"

Then the resulting JSON object is:

```
{
  "isSuccessful": true,
  "resultSet": [{
    "fromAccount": "12345",
    "toAccount": "54321",
    "transactionAmount": 180.00,
    "transactionDate": "2009-03-11T11:08:39.000Z",
    "transactionId": "W06091500863",
    "transactionType": "Funds Transfer"
  }, {
    "fromAccount": "12345",
    "toAccount": null,
    "transactionAmount": 130.00,
    "transactionDate": "2009-03-07T11:09:39.000Z",
    "transactionId": "W214122\/5337",
    "transactionType": "ATM Withdrawal"
  }]
}
```

- The **isSuccessful** property defines whether the invocation was successful.
- The **resultSet** object is an array of returned records.
  - To access the **resultSet** object on the client-side, write:
    `result.invocationResult.resultSet`
  - To access the **resultSet** object on the server-side, write:
    `result.ResultSet`

JavaScript server-side API:

JavaScript adapters can use the IBM MobileFirst Platform Server JavaScript API to perform server-related operations such as: calling other adapters, logging adapter activity, getting values of configuration properties, reporting activities to IBM MobileFirst Analytics, and getting the identity of the request issuer.

MFP.Server and MFP.Logger

The JavaScript server-side API is provided in two classes:
- MFP.Server: for server operations
- MFP.Logger: for logging

Examples of the use of the API are provided in the following sections.

Security

The MFP.Server.getTokenIntrospectionData function provides access to the security context of the client and the client registration data. The following sample uses the function to get the display name of the authenticated user:

```
AuthenticatedUser user =
    securityContext.getAuthenticatedUser();
    return "Hello " + user.getDisplayName();
}
```

Calling adapter procedures

The MFP.Server.invokeProcedure makes it easy to perform requests to other adapters in the same server.

The following example shows how to use this function to call a JavaScript adapter:

```
function callAnotherProcedure() {
    var invocationData = {
        adapter : "JsAdapter", procedure :
        "getStories"
    };
return MFP.Server.invokeProcedure();}
```

For more information, see “Configuring adapters” on page 7-227.

Configuration properties

The MFP.Server.getPropertyValue function enables the adapter to read a property from the adapter configuration.

For example, assume you have a user-defined property, databaseName. To get its value, you could write the following code:

```
var dbName = MFP.Server.getPropertyValue('databaseName');
```

For more information, see “Configuring adapters” on page 7-227.

Analytics

The MFP.Server.logActivity function reports information to IBM MobileFirst Analytics.

For example, to send the string `Getting account balance`, you might write:

```
function getBalance(user) {
    MFP.Server.logActivity('Getting account balance);
    // perform operation
}
```

Logging

The JavaScript API provides logging capabilities through the MFP.Logger class. It contains four functions that correspond to four standard logging levels.

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. For example:
   ```
   var x = new MyJavaClass();
   var y = x.myMethod(1, "a");
   ```
3. Add the Java classes in either of the following ways:
   - As source files of the JavaScript adapter, under `<`*`adapter`*`>/src/main/java/<`*`package`*`>`.
   - As Maven dependencies. See “Third-party Maven dependencies” on page 7-192.

Example

The following snippets demonstrate how to invoke custom Java classes from the JavaScript code in your adapter. Assume you are adding a class Calculator to your Java class and that it contains a static method addTwoIntegers and an instance method subtractTwoIntegers:

```
public class Calculator {
    // Add two integers
    public static int addTwoIntegers (int first, int second){
        return first+second ;
    }

    // Subtract two integers
    public int subtractTwoIntegers (int first, int second){
        return first-second ;
    }
}
```

Invoke the static Java method and use the full class name to reference it directly

```
function addTwoIntegers (a, b){
    return {
        result: com.sample.customcode.Calculator.addTwoIntegers (a, b)
        };
}
```

Use the instance method: create a class instance and invoke the instance method from it

```
function subtractTwoIntegers (a, b){
    var calcInstance = new com.sample.customcode.Calculator();
    return {
        result: calcInstance.subtractTwoIntegers (a, b)
    };
}
```

Configuring adapters

Learn how you can override properties during run time, without having to redeploy adapters.

About this task

Starting with V8.0.0 of MobileFirst Server, administrators can use the MobileFirst Operations Console to modify the behavior of an adapter that has been deployed. After configuration has been modified, the changes take effect in the server immediately, without the need to redeploy the adapter, or restart the server. For more information, see “Configuring adapter properties with MobileFirst Operations Console” on page 7-228 in this page.

There are two levels of properties that can be modified on-the-fly:

- In both SQL and HTTP JavaScript adapters, you can configure the *predefined* properties that relate to the connection policy. For a full description of all these properties, see “HTTP adapter connectionPolicy element” on page 7-209 and “SQL adapter connectionPolicy element” on page 7-213.
- In all adapters, you can also configure *user-defined* properties. For more information about creating user-defined properties, see “Creating user-defined adapter properties” on page 7-228 in this page.

Creating user-defined adapter properties

About this task

You use a `property` element for each user-defined property you want to add. For more information on the `property` element, see property element of the JavaScript adapter XML file, or property element of the Java adapter XML file.

Procedure

1. Open the adapter-descriptor (`adapter.xml`) file for your adapter in an editor.
2. For each new property, add a `property` element to the file, as follows:

   ```
   <property name="unique-name"
       description="value"
       defaultValue="value"
       type="value"
   />
   ```

   Note: If the `adapter.xml` file of a JavaScript adapter contains `procedure` elements, the `property` elements must always appear **below** them.
3. Save the `adapter.xml` file.
4. To apply your changes and make your custom properties available in the MobileFirst Server, build your adapter and deploy it to an instance of the server.

Results

After you successfully deploy an adapter with custom properties to the server, you make the value of the properties available to the adapter code with a call to the appropriate API:

- For JavaScript: MFP.Server.getPropertyValue (propertyName)
- For Java: ConfigurationAPI.getPropertyValue (propertyName)

See "Using user-defined property values in adapter code" on page 7-229 in this page.

Configuring adapter properties with MobileFirst Operations Console

About this task

Both the built-in connection policy properties as well as the user-defined properties can be overridden in the MobileFirst Operations Console.

Example

Assume that you have deployed a JavaScript adapter JavaSQL to MobileFirst Server. Assume that the `adapter.xml` descriptor file contains three user-defined properties, as follows:

```
<mfp:adapter name="JavaSQL"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:sql="http://www.ibm.com/mfp/integration/sql">

    <displayName>JavaSQL</displayName>
    <description>JavaSQL</description>
    <connectivity>
        <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>
    </connectivity>

     <!-- Procedures -->
     <procedure name="procedure1"/>

    <!-- Custom properties -->
    <property name="DB_url" displayName="Database URL" defaultValue="jdbc:mysql://127.0.0.1:3306/mobilefirst_trainin
    <property name="DB_username" displayName="Database username" defaultValue="mobilefirst"  />
    <property name="DB_password" displayName="Database password" defaultValue="mobilefirst"  />
</mfp:adapter>
```

You can view the configuration settings by clicking the **Configurations** tab under **mfp Runtime** > *adapter_name*. The predefined connection policy parameters are displayed under **Connectivity**. Beneath them, under **Parameters**, are the user-defined properties settings. In this example, **Database URL**, **Database username**, and **Database password**.

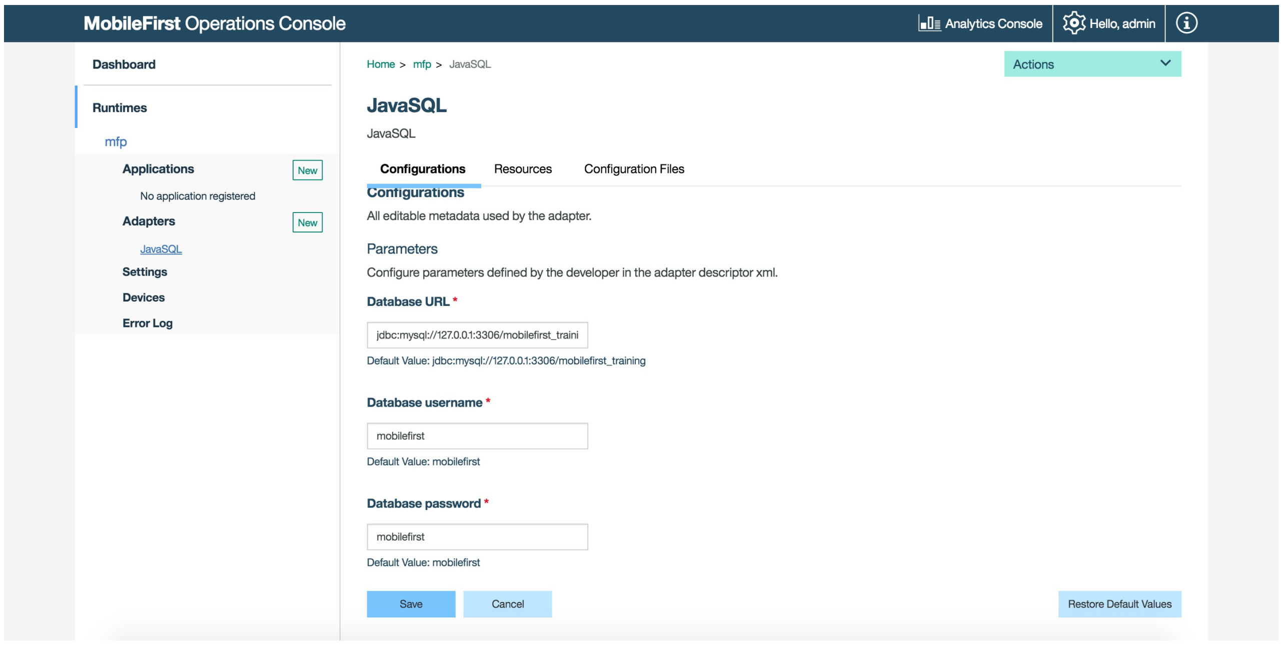

MobileFirst Operations Console
Analytics Console
Hello, admin
Dashboard
Runtimes
mfp
Applications
New
No application registered
Adapters
New
JavaSQL
Settings
Devices
Error Log
Home > mfp > JavaSQL
Actions
JavaSQL
JavaSQL
Configurations
Resources
Configuration Files
Configurations
All editable metadata used by the adapter.
Parameters
Configure parameters defined by the developer in the adapter descriptor xml.
Database URL *
jdbc:mysql://127.0.0.1:3306/mobilefirst_traini
Default Value: jdbc:mysql://127.0.0.1:3306/mobilefirst_training
Database username *
mobilefirst
Default Value: mobilefirst
Database password *
mobilefirst
Default Value: mobilefirst
Save
Cancel
Restore Default Values

Figure 7-12. User-defined adapter properties in the MobileFirst Operations Console

Administrators can modify the values that are shown and save. The changes take effect immediately, without redeploying the adapter.

Sharing adapter configurations

About this task

Customized adapter properties appear in the modified adapter configuration file in the **Configuration files** tab of MobileFirst Operations Console. Use the **configpull** and **configpush** goals to share the custom configuration. See Maven adapter plug-in.

Using user-defined property values in adapter code

About this task

There are both Java and JavaScript server-side APIs that enable you to retrieve properties defined in the `adapter.xml` file or in MobileFirst Operations Console.

- In Java, use the ConfigurationAPI class. Inside your Java class, add the following at the class level:

  ```
  @Context
  ConfigurationAPI configurationAPI ;
  ```

  Then you can use the `configurationAPI` instance to get properties:

  ```
  configurationAPI.getPropertyValue ("DB_url");
  ```

  When the adapter configuration is modified from the MobileFirst Operations console, the JAX-RS application class is reloaded and its `init` method is called again.

  Note: The `getServerJNDIProperty` method can also be used to retrieve a JNDI property from your server configuration.
- In JavaScript, use the MFP.Server.getPropertyValue(propertyName) function to retrieve properties:

  ```
  MFP.Server.getPropertyValue("name");
  ```

Tools for testing and debugging adapters

MobileFirst Java adapters expose a full REST API that enables you to test functionality by issuing HTTP requests.

You can test adapters using MobileFirst Platform CLI, as well as third-party tools such as Swagger and Postman.

The base URL for an adapter is:

`<`*`server-address`*`>/<`*`context-root`*`>/api/adapters/<`*`adaptername`*`>`.

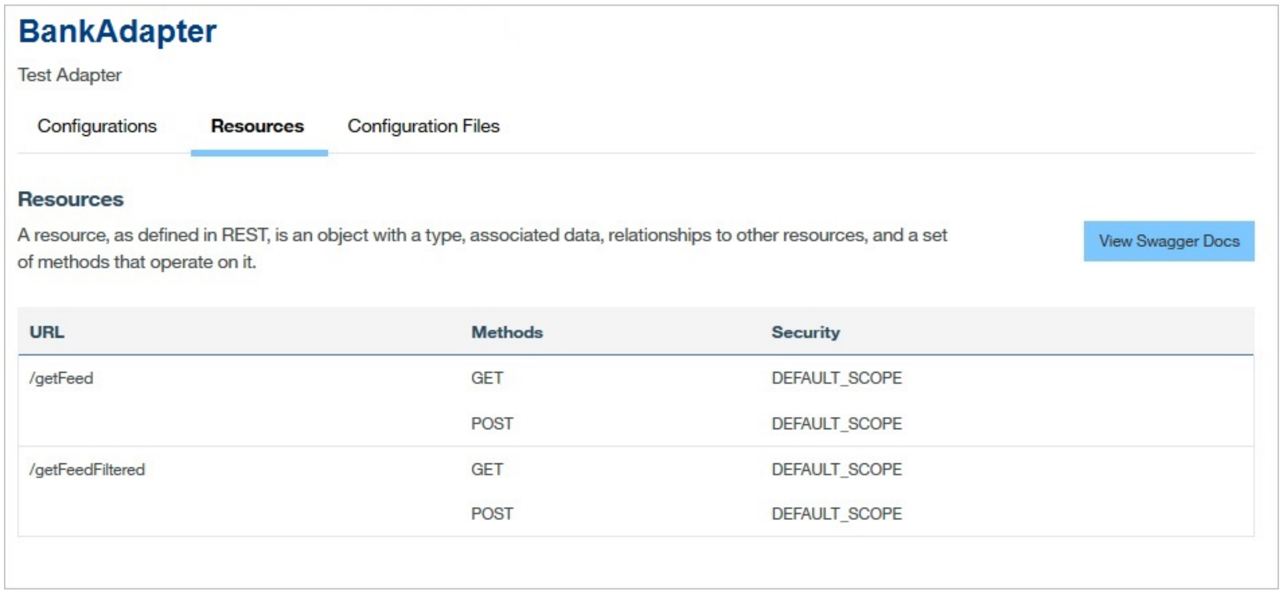
Testing adapters with MobileFirst Platform CLI

To test an adapter by using CLI, you call it by running the **`mfpdev adapter call`** command.

For more information on the command, see “Command-line interface (CLI) summary” on page 7-17, or run the **`mfpdev help adapter call`** command.

Testing adapters with Swagger UI

MobileFirst Development Server ships with a built-in Swagger UI which displays a graphical representation of the Swagger document for the endpoints that are exposed by adapters. From the MobileFirst Operations Console, display the Swagger UI by clicking **View Swagger Docs** in the **Resources** tab of the adapter.


BankAdapter

Test Adapter

Resources

A resource, as defined in REST, is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

URL	Methods	Security
/getFeed	GET	DEFAULT_SCOPE
	POST	DEFAULT_SCOPE
/getFeedFiltered	GET	DEFAULT_SCOPE
	POST	DEFAULT_SCOPE

Figure 7-13. Viewing a Swagger Document

Swagger-UI provides a convenient way of testing adapters, by letting you specify the appropriate parameters for every adapter endpoint. In addition, it is able to implicitly obtain a token for any scope that is used by the adapter, thus avoiding the need to manually obtain the token.

Testing adapters with external Swagger tools

MobileFirst Server exposes an endpoint that provides Swagger 2.0 documentation so that you can test adapters with any tool that parses Swagger 2.0 JSON format. The URL of the endpoint is:`<server-ip>:<server-port>/<context-root>/api/adapterdoc/<adaptername>`

Testing adapters with REST clients such as Postman

You can use Postman or similar tools to test HTTP requests and pass the following parameters:

- URL parameters
- Path parameters
- Body parameters
- Headers

If your resource is protected by a security scope, perform the necessary steps for acquiring an access token and accessing the protected resource. See “Acquiring access tokens” on page 7-280 and “Accessing protected resources” on page 7-281.

Debugging Java adapters

You debug the Java code in an adapter just as you do in any remote Java debug. Connect to MobileFirst Server on the debug port using your favorite IDE. The debug port varies from server to server. In a MobileFirst Server that is based onWebSphere® Application Server Liberty profile, it is 10777 by default.

Client access to adapters

Mobile clients can access both Java and JavaScript adapters from the `/adapters` endpoint on the server.

The URL pattern for accessing the `/adapters` endpoint is as follows:

```
http(s)://<server>:<port>/<Context>/api/adapters/<adapter-name>/*
```

For example, assuming that `http://mfp-server-host/project` is the IBM MobileFirst Platform Foundation 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/api/adapters/adapter1/res1
```

Note: Using the `/adapters` endpoint is the recommended way to access IBM MobileFirst Platform Foundation adapters. This endpoint supports both JavaScript and Java adapters and is protected by an OAuth security mechanism.

Accessing adapters from a mobile client

IBM MobileFirst Platform Foundation 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 examples demonstrate how to use the client API to access an adapter resource:

Cordova JavaScript client

```
var request = new WLResourceRequest("adapters/adapter1/res1", WLResourceRequest.GET);
    request.send().then(
        function(response) {
            alert(JSON.stringify(response));
        },
    function(error) {
        alert(JSON.stringify(error));
    }
);
```

Native Android client

```
WLResourceRequest req = new WLResourceRequest(new URI("adapters/adapter1/res1"), WLResourceRequest.GET);
req.send(new WLResponseListener(){
    @Override
    public void onSuccess(WLResponse response) {
        // handle success

    }
    @Override
    public void onFailure(WLFailResponse response) {
        // handle failure

    }});
```

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]];
    }
}];
```

Native Windows 10 UWP and Windows 8 Universal

```
WLResourceRequest req = new WLResourceRequest("adapters/adapter1/res1", WLResourceRequest.GET);
InvokeListener listener = new InvokeListener();
req.send(listener);
```

```
public class InvokeListener : WLResponseListener {
    public void onSuccess(WLResponse response){
        //handle success
    }

    public void onFailure(WLFailResponse response){
        //handle failure
    }
}
```

RESTful access to Java adapters

You call an existing Java adapter over HTTP via REST URLs.

The URL pattern is as follows:

```
http(s)://<server>:<port>/<Context>/adapters/<adapter-name>/*
```

For example:

```
http://<hostname>:<port>/mfp/adapters/TodaysNews/getStory?story=world
```

Java adapters support all REST features, so you can use all HTTP request methods, headers, query parameters, and more.

RESTful access to JavaScript adapters

You call existing JavaScript adapter procedures over HTTP via REST URLs.

The URL pattern is as follows:

```
http(s)://<server>:<port>/<Context>/api/adapters/<adapter-name>/<procedure-name>
```

For example:

```
http://<hostname>:<port>/mfp/api/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).

Troubleshooting an error when an application or an adapter is pushed to a MobileFirst Server

Symptoms

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**.

Causes

All applications and adapters are pushed to a MobileFirst Server runtime. The runtime is defined by the MobileFirst server that is running. Before you push these applications and adapters to the MobileFirst Server, make sure the server is running. The error that you encountered indicates that the CLI cannot locate the associated runtime on the target MobileFirst Server.

The CLI cannot detect a runtime on a MobileFirst Server in the following cases:
- The MobileFirst Server is not running.
- No network connection is established between the system that runs the CLI and the MobileFirst Server.
- You have not deployed the runtime to the MobileFirst Server.

Resolving the problem

1. Confirm that the MobileFirst Server is running.

   Local MobileFirst Server

   a. Start your MobileFirst Server if it is not running.

   Remote MobileFirst Server

   a. Run the **`mfpdev 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 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 no network connectivity issues exist.
   c. Assuming the server is running, run the **`mfpdev server info <server>`** command, where `<server>` is the `Name` value from the previous **`mfpdev 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, enter `mfpdev help server info` on the IBM MobileFirst Platform Foundation command line interface.
2. 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 **`mfpdev server info`** command and identify the target remote server `Name` value. The local MobileFirst Server name is typically `local`.
   b. Run the **`mfpdev server info <server>`** command, where *<server>* is the `Name` value from step 2a.
   c. Look at the command output and find the runtimes listed under the `Runtime` section.
   d. Determine whether you are pushing to one of the runtimes that are listed in step 2c by looking at the log output of the push command. The log output should include the following information:
      ```
      Pushing app_name_or_adapter_name to server: 'server_url'
      runtime:'runtime_name'.
      ```
   e. If you are not using one of the runtimes in step 2c, complete one of the following steps:
      - Specify the name of the target runtime when you run the **`mfpdev app push`** command. Run the `mfpdev help app push` command for more information about how to specify the name of the runtime.
      - Change the runtime name in your app config by entering the following command:
        ```
        mfpdev app config runtime runtime_name
        ```

Updating Cordova client apps directly

With direct updates, you deliver updated web resources directly to deployed client applications.

Subject to the terms and conditions of the target platform, organizations are not required to upload new app versions to the app store or market. By using the Direct Update feature, you can quickly update application web resources (HTML, JavaScript, and CSS) without going through the vendor (Apple/Google) app store review process.

Direct Update is activated automatically when web resources are deployed in the MobileFirst Server. Once activated, it will be enforced on every request to a protected resource. When you use the Direct Update feature and the web resources checksum feature is enabled, a new checksum base is established with each Direct Update.

Direct Update is not intended for updating native code.

Supported platforms

Direct Update is available only for iOS and Android Cordova apps.

Prerequisites

If the MobileFirst Server was upgraded by using a fix pack, it continues to serve direct updates properly. However, if a recently built Direct Update archive (.zip file) is uploaded, it can halt updates to older clients. The reason is that the archive contains the version of the `cordova-plugin-mfp` plug-in. Before it serves that archive to a mobile client, the server compares the client version with the plug-in version. If both versions are close enough (meaning that the three most significant digits are identical), Direct Update occurs normally. Otherwise, MobileFirst Server silently skips the update. One solution for the version mismatch is to download the `cordova-plugin-mfp` with the same version as the one in your original Cordova project and regenerate the Direct Update archive.

Incremental and full Direct Update

Client applications built on IBM MobileFirst Platform Foundation V8.0.0:

- Receive an *incremental* update if the web resources of the application are only *one* build behind those in the application that is now being deployed. Only the web resources that were changed since the last deployment are downloaded and updated.
- Receive a *full* update if the web resources of the application are more than one build behind those in the application that is now being deployed.

Secure and non-secure Direct Update

For secure Direct Update to work, deploy a user-defined keystore file in MobileFirst Server and include a copy of the matching public key in the client application. If the client is not configured with a public key, MobileFirst Server uses the default server keystore to sign Direct Update but does not enforce signature matching.

If secure Direct Update was enabled and the archive signature was compromised, the client halts the update. Failures such as these are reported in the server logs.

Note: Failures such as these might also prevent correct adapter invocation.

For more information about implementing secure Direct Update, see "Implementing secure Direct Update on the client side" on page 7-239.

Direct Update in development, testing, and production

For development and testing purposes, developers typically use Direct Update by simply uploading an archive to the development server. While this process is easy to implement, it is not very secure. For this phase, an internal RSA key pair that is extracted from an embedded MobileFirst self-signed certificate is used.

For the phases of live production or even pre-production testing, however, it is strongly recommended to implement secure Direct Update before you publish your application to the app store. Secure Direct Update requires an RSA key pair that is extracted from a real CA signed server certificate.

To implement secure Direct Update, deploy a user-defined keystore to the MobileFirst Server and copy the matching public key to the client application. For more information about implementing secure Direct Update, see "Implementing secure Direct Update on the client side" on page 7-239.

Note: Take care that you do not modify the keystore configuration after the application was published, updates that are downloaded can no longer be authenticated without reconfiguring the application with a new public key and republishing the application. Without performing these two steps, Direct Update fails on the client.

For more information, see "The Direct Update lifecycle."

Direct Update data transfer rates

At optimal conditions, a single MobileFirst Server can push data to clients at the rate of 250 MB per second. If higher rates are required, consider a cluster or a CDN service.

For more information, see "Serving Direct Update requests from a CDN" on page 7-241.

The Direct Update lifecycle

Package and upload updated web resources to the MobileFirst Server. Deployed client Cordova apps can then download them with Direct Update.

A Direct Update scenario

The following diagram shows a typical production scenario. Web resources of a Cordova application that was published to an app store and downloaded to users' devices are updated directly.

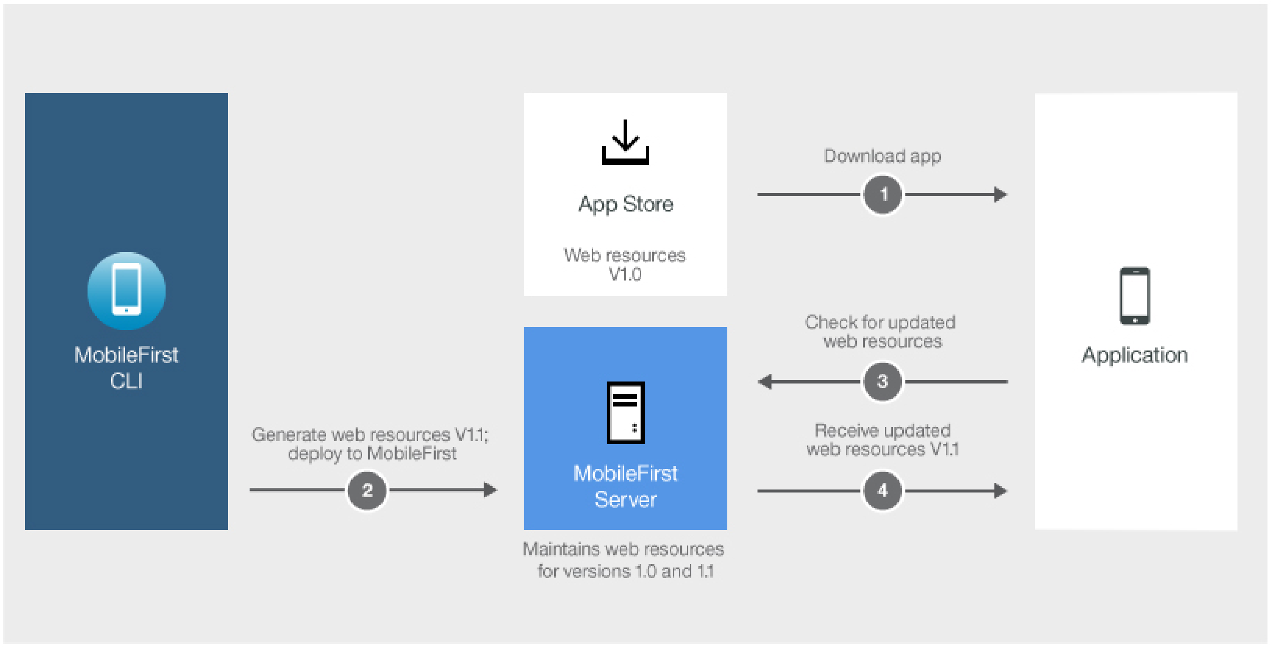

MobileFirst CLI
App Store
Web resources V1.0
Download app
1
Application
Check for updated web resources
3
Generate web resources V1.1; deploy to MobileFirst
2
MobileFirst Server
Receive updated web resources V1.1
4
Maintains web resources for versions 1.0 and 1.1

Figure 7-14. Direct update cycle

1. The user downloads the application that was published in the app store. The application is marked and is published as V1.0.
2. Using CLI, the developer generates a package of updated web resources and deploys it to MobileFirst Server
3. Each time the client application makes a resource request to the MobileFirst Server, it checks for updates to the web resources. If any are found, the client is notified and is prompted to download the modified files, for example, a new HTML file.
4. After the download finishes, the general version of the application as shown on the user's device and as shown in the MobileFirst Operations Console is V1.0; however, internally, the web resources that it contains were revised slightly (call it V1.1). The only evidence that something changed is the application checksum, in addition to several internal files that changed too.

Creating and deploying updated web resources to MobileFirst Server

To make modified web resources available to deployed client Cordova applications, you package and upload them as an archive (.zip file) to the MobileFirst Server.

About this task

You use either MobileFirst Platform CLI or MobileFirst Operations Console to perform procedures on the updated web resources package:

- You can use MobileFirst Platform CLI commands to generate and upload the archive.
- You can use the MobileFirst Operations Console only to upload an existing archive.

Creating and deploying updated web resources to the default MobileFirst Server

Procedure

1. From the command line, navigate to the root of the Cordova project.
2. Run the command:
   ```
   mfpdev app webupdate
   ```
   The updated web resources are packaged to an archive and uploaded to the default MobileFirst Server that is running in the developer workstation. The packaged web resources are located in the *<cordova-project-root-folder>*/ `mobilefirst/` folder.

   The archive in the `mobilefirst` folder contains important metadata. It embeds the version of the `cordova-plugin-mfp` plug-in. Before it serves that archive to a mobile client, the server compares the client version with the plug-in version. If both versions are close enough (meaning that the three most significant digits are identical), Direct Update occurs normally. Otherwise, MobileFirst Server silently skips the update.

Creating and deploying updated web resources to a non-default MobileFirst Server

Procedure

1. Build the archive.
2. Run the command:
   ```
   mfpdev app webupdate [server-name] [runtime-name]
   ```
   For example:
   ```
   mfpdev app webupdate myQAServer MyBankApps
   ```

Uploading a previously generated archive

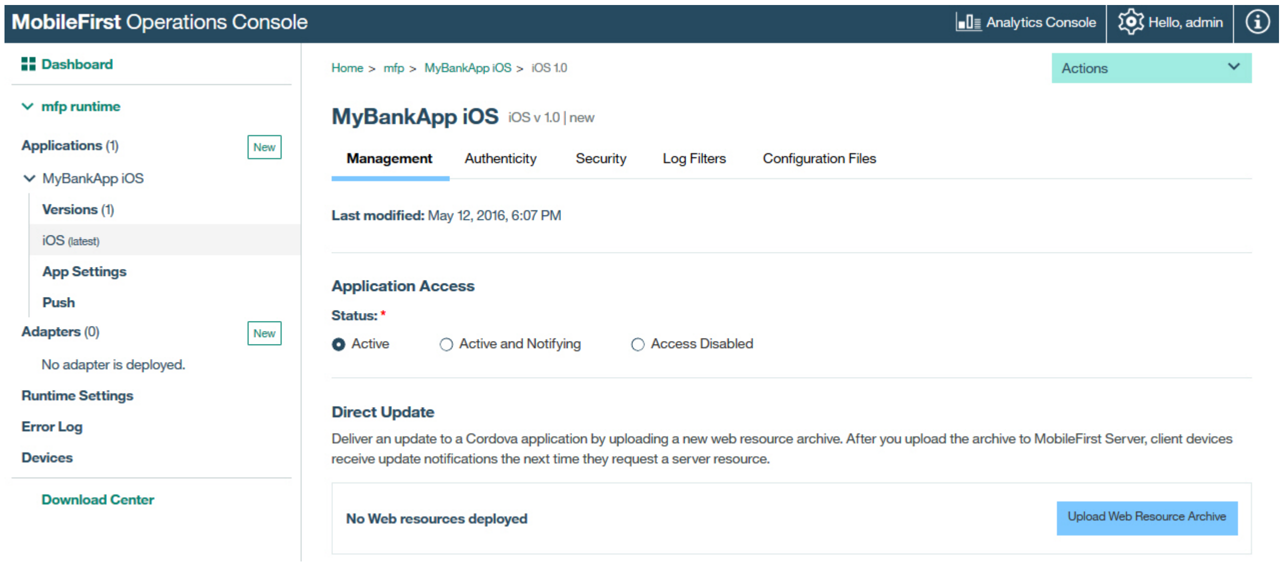
Procedure

Run the command:
```
mfpdev app webupdate [server-name] [runtime-name] --file [path-to-packaged-web-resources]
```
For example:
```
mfpdev app webupdate myQAServer MyBankApps --file mobilefirst/ios/com.mfp.myBankApp-1.0.1.zip
```

Uploading packaged web resources with the MobileFirst Operations Console

Procedure

1. Build the archive.
   ```
   mfpdev app webupdate --build
   ```
2. Display the MobileFirst Operations Console from the following URL:
   ```
   http://localhost:9080/mfpconsole/
   ```



Last modified: May 12, 2016, 6:07 PM

Application Access

Status: *

[✓ Active] Active and Notifying Access Disabled

Direct Update

Deliver an update to a Cordova application by uploading a new web resource archive. After you upload the archive to MobileFirst Server, client devices receive update notifications the next time they request a server resource.

No Web resources deployed

Figure 7-15. MobileFirst Operations Console for uploading an archive

3. Click an application version in the navigation sidebar.
4. Click **Upload Web Resources Archive** and in the file chooser that is displayed, choose the archive for upload.

Implementing secure Direct Update on the client side

You can configure client applications to validate the authenticity of a Direct Update package that is downloaded from the MobileFirst Server or CDN.

About this task

For secure Direct Update to work, a user-defined keystore file must be deployed in MobileFirst Server and a copy of the matching public key must be included in the deployed client application. For more information, see Secure and non-secure Direct Update.

This topic describes how to bind a public key to new client applications and existing client applications that were upgraded. For more information on configuring the keystore in MobileFirst Server, see “Configuring the MobileFirst Server keystore” on page 7-316.

The server provides a built-in keystore that can be used for testing secure Direct Update for development phases.

Note: After you bind the public key to the client application and rebuild it, you do not need to upload it again to the MobileFirst Server. However, if you previously published the application to the market, without the public key, you must republish it.

For development purposes, the following default, dummy public key is provided with MobileFirst Server:

```
-----BEGIN PUBLIC KEY-----
MIIDPjCCAiagAwIBAgIEUD3/bjANBgkqhkiG9w0BAQsFADBgMQswCQYDVQQGEwJJTDELMAkGA1UECBMCSUwxETA
PBgNVBAcTCFNoZWZheW1tMQwwCgYDVQQKEwNJQk0xEjAQBgNVBAsTCVdvcmtsaWdodDEPMA0GA1UEAxMGV0wgRG
V2MCAXDTEyMDgyOTExMzkyNloYDzQ3NTAwNzI3MTEzOTI2WjBgMQswCQYDVQQGEwJJTDELMAkGA1UECBMCSUwxE
TAPBgNVBAcTCFNoZWZheW1tMQwwCgYDVQQKEwNJQk0xEjAQBgNVBAsTCVdvcmtsaWdodDEPMA0GA1UEAxMGV0wg
```

```
RGV2MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzQN3vEB2/of7KAvuvyoIt0T7cjaSTjnOBm0N3+q
zx++dh92KpNJXj/a3o4YbwJXkJ7jU8ykjCYvjXRf0hme+HGhiIVwxJo54iqh76skDS5m7DaseFdndZUJ4p7NFVw
I5ixA36ZArSZ/Pn/ej56/RRjBeRI7AEGXUSGojBUPA6J6DYkwaXQRew9l+Q1kj4dTigyKL5Os0vNFaQyYu+bT2E
vnOixQ0DXm94IqmHZamZKbZLrWcOEfuAsSjKYOdMSM9jkCiHaKcj7fpEZhUxRRs7joKs1Ri4ihs6JeUvMEiG4gK
l9V3FP/Huy0pfkL0F8xMHgaQ4c/lxS/s3PV0OEg+7wIDAQABMA0GCSqGSIb3DQEBCwUAA4IBAQAgEhhqRl2Rgkt
MJeqOCRcT3uyr4XDK3hmuhEaE0nOvLHi61PoLKnDUNryWUicK/W+tUP9jkN5xRckdzG6TJ/HPySmZ7Adr6QRFu+
xcIMY+/S8j4PHLXBjoqgtUMhkt7S2/thN/VA6mwZpw4Ol0Pa2hyT2TkhQoYYkRwYCk9pxmuBCoH/eCWpSxquNny
RwrY25x0YzccXUaMI8L3/3hzq3mW40YIMiEdpiD5HqjUDpzN1funHNQdsxEIMYsWmGAwOdV5slFzyrH+ErUYUFA
pdGIdLtkrhzbqHFwXE0v3dt+lnLf21wRPIqYHaEu+EB/A4dLO6hm+IjBeu/No7H7TBFm
-----END PUBLIC KEY-----
```

Important: Do not use the public key for production purposes. See “Direct Update in development, testing, and production” on page 7-236.

There are many tools available for generating certificates and extracting public keys from a keystore. The following example demonstrates the procedures with the JDK keytool utility and openSSL.

Procedure

1. Extract the public key from the keystore file that is deployed in the MobileFirst Server.

   Note: The public key must be Base64 encoded.
   For example, assume that the alias name is mfp-server and the keystore file is `keystore.jks`.

   a. To generate a certificate, issue the following command:
   ```
   keytool -export -alias mfp-server -file certfile.cert
    -keystore keystore.jks -storepass keypassword
   ```
   A certificate file is generated.

   b. Issue the following command to extract the public key:
   ```
   openssl x509 -inform der -in certfile.cert -pubkey -noout
   ```
   Note: Keytool alone cannot extract public keys in Base64 format.

2. Perform one of the following procedures:
   - Copy the resulting text, without the `BEGIN PUBLIC KEY` and `END PUBLIC KEY` markers into the `mfpclient` property file of the application, immediately after `wlSecureDirectUpdatePublicKey`.
   - From the command prompt, issue the following command:
     ```
     mfpdev app config direct_update_authenticity_public_key <public_key>
     ```
     For <*public_key*>, paste the text that results from Step 1, without the `BEGIN PUBLIC KEY` and `END PUBLIC KEY` markers.

3. Run the **`cordova build`** command to save the public key in the application.

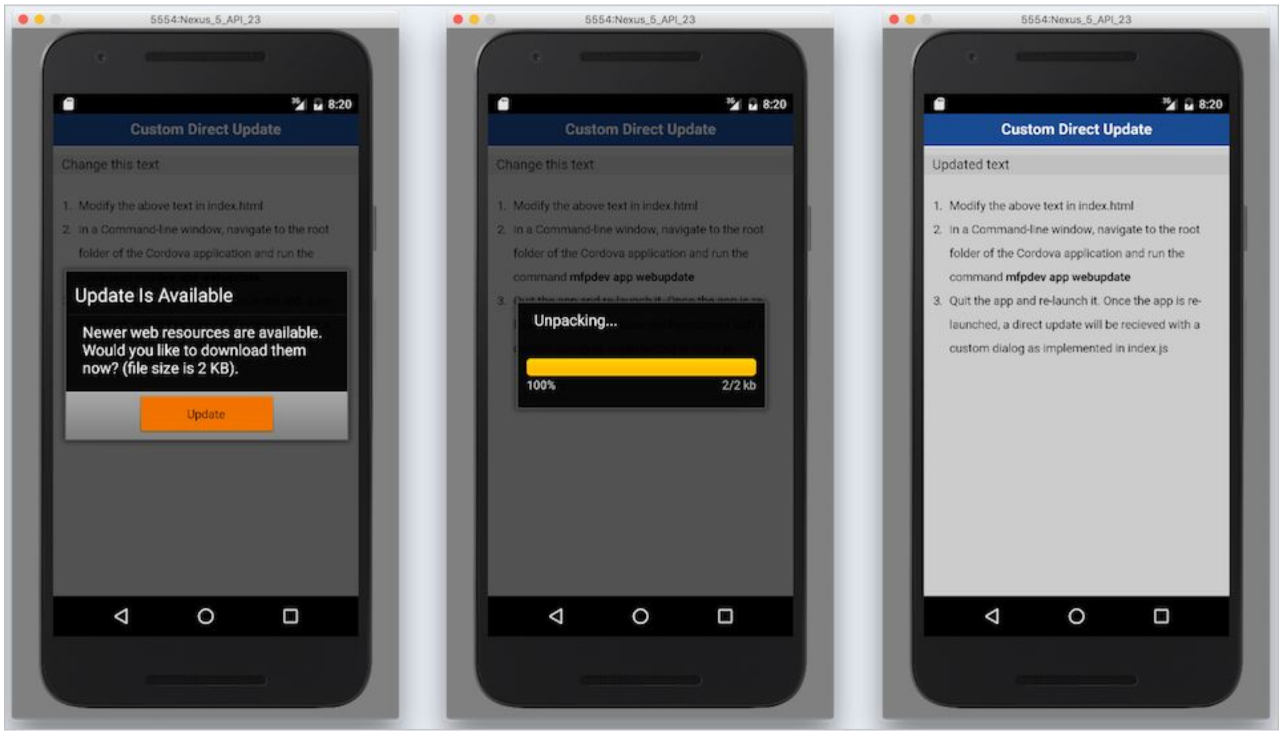
Default Direct Update user interface

Learn about the default user interface of the Direct Update feature.

Default UI for Direct Update

When the app receives an update from the MobileFirst Server, it starts downloading the newly deployed resources, as shown in the following figures. If the download fails mid-way, the direct update resumes from where the download was broken the previous time.

The user notifications in the following figures show the default method of implementing Direct Update.



Update Is Available

Newer web resources are available. Would you like to download them now? (file size is 2 KB).

1. Modify the above text in index.html
2. In a Command-line window, navigate to the root folder of the Cordova application and run the command **mfpdev app webupdate**
3. Quit the app and re-launch it. Once the app is re-launched, a direct update will be recieved with a custom dialog as implemented in index.js

Figure 7-16. Default Direct Update notices

Serving Direct Update requests from a CDN

You can configure Direct Update requests to be served from a CDN (content delivery network) instead of from the MobileFirst Server.

Advantages of using a CDN

Using a CDN instead of the MobileFirst Server to serve Direct Update requests has the following advantages:

- Removes network overheads from the MobileFirst Server.
- Increases transfer rates higher than the 250 MB/second limit when serving requests from a MobileFirst Server.
- Ensures a more uniform Direct Update experience for all users regardless of their geographical location.

General requirements

To serve Direct Update requests from a CDN, ensure that your configuration conforms to the following conditions:

- The CDN must be a reverse proxy in front of the MobileFirst Server (or in front of another reverse proxy if needed).
- When building the application from your development environment, set up your target server to the CDN host and port instead of the host and port of the MobileFirst Server. For example, when running the MobileFirst Platform CLI command `mfpdev server add`, provide the CDN host and port.

- In the CDN administration panel, you need to mark the following Direct Update URLs for caching to ensure that the CDN passes all requests to the MobileFirst Server except for the Direct Update requests. For Direct Update requests, the CDN determines whether it obtained the content. If it has, it returns it without going to the MobileFirst Server; if not, it goes to the MobileFirst Server, gets the Direct Update archive (.zip file), and stores it for the next requests for that specific URL. For applications that are built with V8.0.0 of IBM MobileFirst Platform Foundation, the Direct Update URL is: `PROTOCOL://DOMAIN:PORT/CONTEXT_PATH/api/directupdate/VERSION/CHECKSUM/TYPE`.

  The `PROTOCOL://DOMAIN:PORT/CONTEXT_PATH` prefix is constant for all runtime requests. For example:`http://my.cdn.com:9080/mfp/api/directupdate/0.0.1/742914155/full?appId=com.ibm.DirectUpdateTestApp&clientPlatform=android`

  In the example, there are additional request parameters that are also part of the request.
- The CDN must allow caching of the request parameters. Two different Direct Update archives might differ only by the request parameters.
- The CDN must support TTL on the Direct Update response. The support is needed to support multiple direct updates for the same version.
- The CDN must not change or remove the HTTP headers that are used in the MobileFirst server-client protocol.

Example configuration

This example is based on using an Akamai CDN configuration that caches the Direct Update archive. The following tasks are completed by the network administrator, the MobileFirst administrator, and the Akamai administrator:

Network administrator

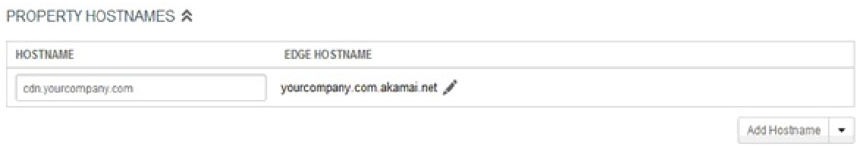
1. Create another domain in the DNS for your MobileFirst Server. For example, if your server domain is yourcompany.com you need to create an additional domain such as cdn.yourcompany.com.
2. In the DNS for the new cdn.yourcompany.com domain, set a CNAME to the domain name that is provided by Akamai. For example, yourcompany.com.akamai.net.

MobileFirst administrator

Set the new cdn.yourcompany.com domain as the MobileFirst Server URL for the MobileFirst applications. For example, for the Ant builder task, the property is: `<property name="wl.server" value="http://cdn.yourcompany.com/${contextPath}/"/>`

Akamai administrator

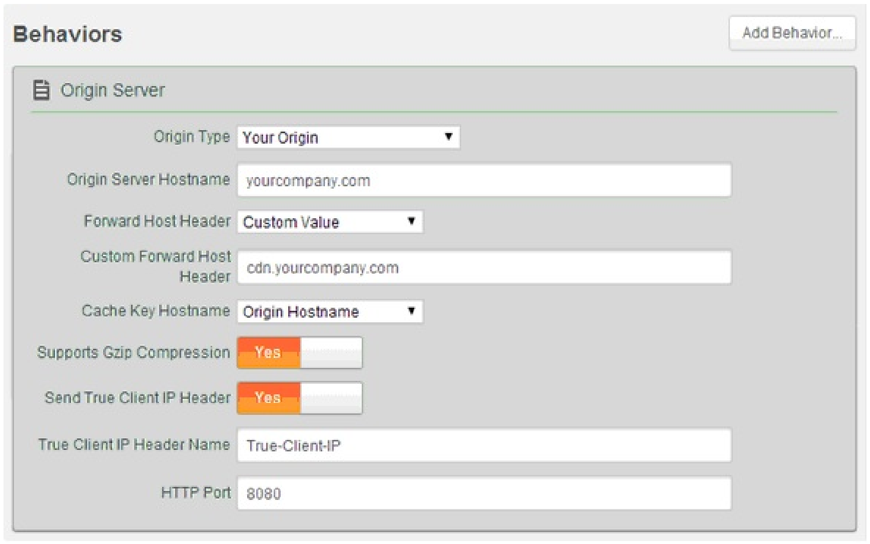
1. Open the Akamai property manager and set the property host name to the value of the new domain.



PROPERTY HOSTNAMES

HOSTNAME	EDGE HOSTNAME
cdn.yourcompany.com	yourcompany.com.akamai.net

2. On the Default Rule tab, configure the original MobileFirst Server host and port, and set the **Custom Forward Host Header** value to the newly created domain.

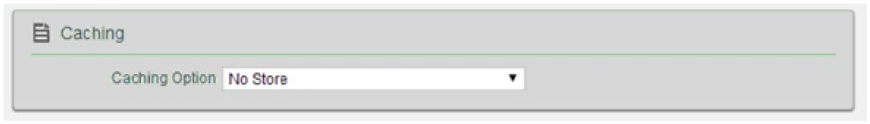


Behaviors

Origin Server

Origin Type	Your Origin
Origin Server Hostname	yourcompany.com
Forward Host Header	Custom Value
Custom Forward Host Header	cdn.yourcompany.com
Cache Key Hostname	Origin Hostname
Supports Gzip Compression	Yes
Send True Client IP Header	Yes
True Client IP Header Name	True-Client-IP
HTTP Port	8080

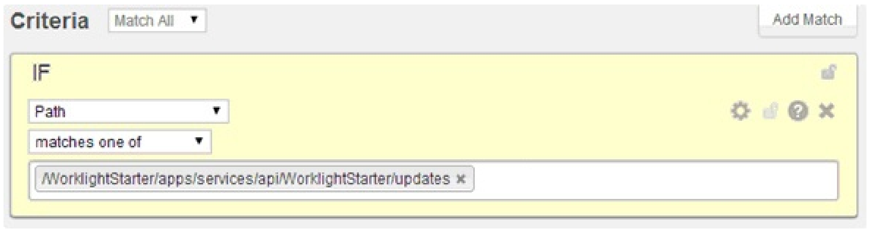
3. From the **Caching Option** list, select `No Store`.



Caching

Caching Option: No Store

4. From the Static Content configuration tab, configure the matching criteria according to the Direct Update URL of the application. For example, create a condition that states `If Path matches one of` *`direct_update_URL`*.



Criteria: Match All

IF

Path

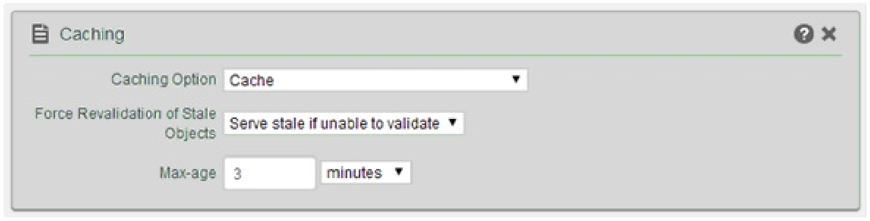
matches one of

/WorklightStarter/apps/services/api/WorklightStarter/updates

5. Set values similar to the following values to configure the caching behavior to make cache the Direct Update URL and to set TTL.

Table 7-20. Configuring caching.

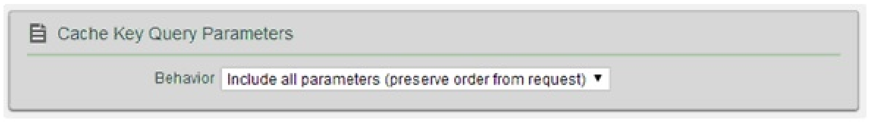
Field	Value
Caching Option	`Cache`
Force Revaluation of Stale Objects	`Serve stale if unable to validate`
Max-age	`3 minutes`



Caching

Caching Option	Cache
Force Revalidation of Stale Objects	Serve stale if unable to validate
Max-age	3 minutes

6. Configure the cache key behavior to use all request parameters in the cache key (you must do so to cache different Direct Update archives for different applications or versions). For example, from the **Behavior** list, select `Include all parameters (preserve order from request)`.



Cache Key Query Parameters

Behavior: Include all parameters (preserve order from request)

7. Save and activate the configuration.

Customizing the Direct Update user interface and process

You can change the default user interface for the Direct Update dialog boxes and the messages that are displayed to the user.

You can control the direct update process without presenting a user interface to the user and control what happens when the direct update process fails.

Override the `handleDirectUpdate` function of the Direct Update challenge handler to customize the direct update process and interface in iOS and Android applications. The `handleDirectUpdate` function is defined inside `worklights.js`. It has the following format:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function (directUpdateData,directUpdateContext){...}
```

The function accepts the following arguments:

directUpdateData
: A JSON object that contains the `downloadSize` property that represents the files size in bytes of the update package to be downloaded from server.

directUpdateContext
: A JavaScript object that exposes a `.start()` and `.stop()` function that start and stop the Direct Update flow.

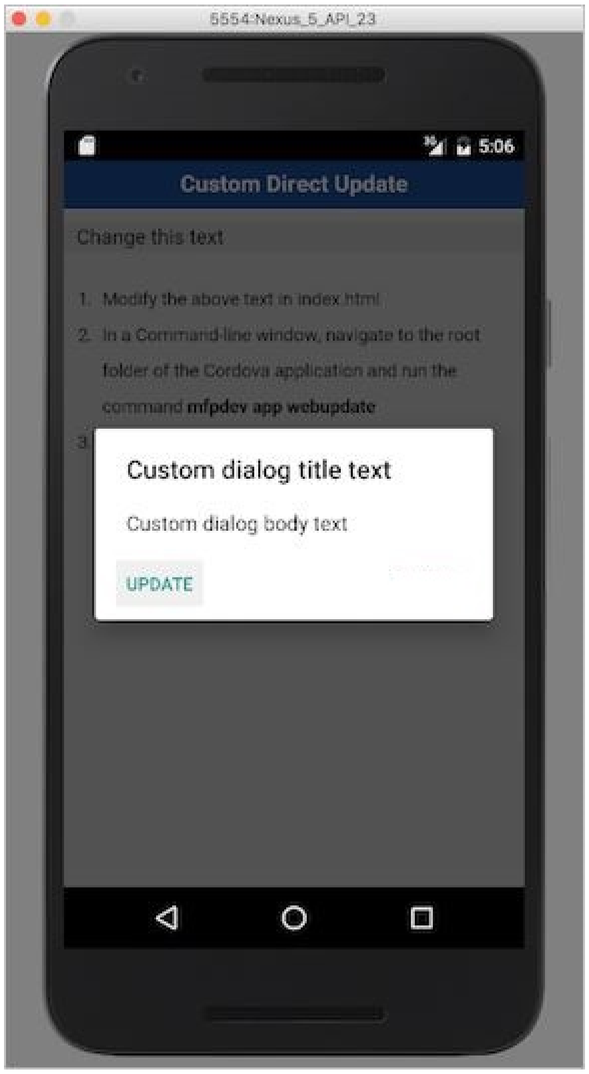
If the web resources are newer on the MobileFirst Server than in the application, Direct Update challenge data is added to the server response. Whenever the MobileFirst client-side framework detects this direct update challenge, it invokes the `wl_directUpdateChallengeHandler.handleDirectUpdate` function.

The function provides a default Direct Update design: a default message dialog that is displayed when a Direct Update is available and a default progress screen that is displayed when the direct update process is initiated. For examples of default screens, see “Default Direct Update user interface” on page 7-240. You can implement custom Direct Update user interface behavior or customize the Direct Update dialog box by overriding this function and implementing your own logic.

The following example `handleDirectUpdate` function implements a custom message in the Direct Update dialog. Add this code into the `www/js/index.js` file of the Cordova project .

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext) {
    navigator.notification.confirm(  // Creates a dialog.
        'Custom dialog body text',
        // Handle dialog buttons.
          directUpdateContext.start();
        },
        'Custom dialog title text',
        ['Update']
    );
};
```

The result is shown in Figure 1.



Custom Direct Update

Change this text

1. Modify the above text in index.html
2. In a Command-line window, navigate to the root folder of the Cordova application and run the command **mfpdev app webupdate**

Custom dialog title text

Custom dialog body text

UPDATE

Figure 7-17. Custom update notice

You can start the direct update process by running the `directUpdateContext.start()` method whenever the user clicks the dialog button. The default progress screen, which resembles the one in previous versions of MobileFirst Server is shown.

This method supports the following types of invocation:

- When no parameters are specified, IBM MobileFirst Platform Server uses the default progress screen.
- When a listener function such as `directUpdateContext.start(directUpdateCustomListener)` is supplied, the direct update process runs in the background while the process sends lifecycle events to the listener. The custom listener must implement the following methods:

```
var  directUpdateCustomListener  = {
    onStart : function ( totalSize ){ },
    onProgress : function ( status , totalSize , completedSize ){ },
    onFinish : function ( status ){ }
};
```

The listener methods are started during the direct update process according to following rules:

- `onStart` is called with the `totalSize` parameter that holds the size of the update file.
- `onProgress` is called multiple times with status `DOWNLOAD_IN_PROGRESS`, `totalSize`, and `completedSize` (the volume that is downloaded so far).
- `onProgress` is called with status `UNZIP_IN_PROGRESS`.
- `onFinish` is called with one of the following final status codes:

Table 7-21. Status codes for the `onFinish` rule

Status code	**Description**
`SUCCESS`	Direct update finished with no errors.
`CANCELED`	Direct update was canceled (for example, because the `stop()` method was called).
`FAILURE_NETWORK_PROBLEM`	There was a problem with a network connection during the update.
`FAILURE_DOWNLOADING`	The file was not downloaded completely.
`FAILURE_NOT_ENOUGH_SPACE`	There is not enough space on the device to download and unpack the update file.
`FAILURE_UNZIPPING`	There was a problem unpacking the update file.
`FAILURE_ALREADY_IN_PROGRESS`	The start method was called while direct update was already running.
`FAILURE_INTEGRITY`	Authenticity of update file cannot be verified.
`FAILURE_UNKNOWN`	Unexpected internal error.

If you implement a custom direct update listener, you must ensure that the app is reloaded when the direct update process is complete and the `onFinish()` method has been called. You must also call `wl_directUpdateChalengeHandler.submitFailure()` if the direct update process fails to complete successfully.

The following example shows an implementation of a custom direct update listener:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
    //show custom progress dialog
  },
  onProgress: function(status,totalSize,completedSize){
    //update custom progress dialog
  },
  onFinish: function(status){

    if (status == 'SUCCESS'){
      //show success message
      WL.Client.reloadApp();
    }
    else {
      //show custom error message

      //submitFailure must be called is case of error
      wl_directUpdateChallengeHandler.submitFailure();
    }
  }
};

wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext){
  WL.SimpleDialog.show('Update Avalible', 'Press update button to download version 2.0', [{
    text : 'update',
    handler : function() {
      directUpdateContext.start(directUpdateCustomListener);
    }
  }]);
};
```

Scenario: Running UI-less direct updates

IBM MobileFirst Platform Foundation supports UI-less direct update when the application is in the foreground.

To run UI-less direct updates, implement `directUpdateCustomListener`. Provide empty function implementations to the `onStart` and `onProgress` methods. Empty implementations cause the direct update process to run in the background.

To complete the direct update process, the application must be reloaded. The following options are available:

- The `onFinish` method can be empty as well. In this case, direct update will apply after the application has restarted.
- You can implement a custom dialog that informs or requires the user to restart the application. (See the following example.)
- The `onFinish` method can enforce a reload of the application by calling `WL.Client.reloadApp()`.

Here is an example implementation of `directUpdateCustomListener`:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
  },
  onProgress: function(status,totalSize,completeSize){
  },
  onFinish: function(status){
    WL.SimpleDialog.show('New Update Available', 'Press reload button to update to new version', [ {
      text : WL.ClientMessages.reload,
      handler : WL.Client.reloadApp
    }]);
  }
};
```

Implement the `wl_directUpdateChallengeHandler.handleDirectUpdate` function. Pass the `directUpdateCustomListener` implementation that you have created as a parameter to the function. Make sure `directUpdateContext.start(directUpdateCustomListener)` is called. Here is an example `wl_directUpdateChallengeHandler.handleDirectUpdate` implementation:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext){

  directUpdateContext.start(directUpdateCustomListener);
};
```

Note: When the application is sent to the background, the direct-update process is suspended.

Scenario: Handling a direct update failure

This scenario shows how to handle a direct update failure that might be caused, for example, by loss of connectivity. In this scenario, the user is prevented from using the app even in offline mode. A dialog is displayed offering the user the option to try again.

Create a global variable to store the direct update context so that you can use it subsequently when the direct update process fails. For example:

```
var savedDirectUpdateContext;
```

Implement a direct update challenge handler. Save the direct update context here. For example:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext){

  savedDirectUpdateContext = directUpdateContext; // save direct update context

  var downloadSizeInMB = (directUpdateData.downloadSize /
1048576).toFixed(1).replace(".", WL.App.getDecimalSeparator());
  var directUpdateMsg = WL.Utils.formatString(WL.ClientMessages.directUpdateNotificationMessage, downloadSizeInMB);

  WL.SimpleDialog.show(WL.ClientMessages.directUpdateNotificationTitle, directUpdateMsg, [{
    text : WL.ClientMessages.update,
    handler : function() {
      directUpdateContext.start(directUpdateCustomListener);
    }
  }]);
};
```

Create a function that starts the direct update process by using the direct update context. For example:

```
restartDirectUpdate = function () {
  savedDirectUpdateContext.start(directUpdateCustomListener);
// use saved direct update context to restart direct update
};
```

Implement `directUpdateCustomListener`. Add status checking in the `onFinish` method. If the status starts with “FAILURE”, open a modal only dialog with the option “Try Again”. For example:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
    alert('onStart: totalSize = ' + totalSize + 'Byte');
  },
  onProgress: function(status,totalSize,completeSize){
    alert('onProgress: status = ' + status + ' completeSize = ' + completeSize + 'Byte');
  },
  onFinish: function(status){
    alert('onFinish: status = ' + status);
    var pos = status.indexOf("FAILURE");
    if (pos > -1) {
      WL.SimpleDialog.show('Update Failed', 'Press try again button', [ {
        text : "Try Again",
        handler : restartDirectUpdate // restart direct update
      }]);
    }
  }
};
```

When the user clicks the **Try Again** button, the application restarts the direct update process.

Push notification

Push notification is the ability of a mobile device to receive messages that are pushed from a server. Notifications are received regardless of whether the application is currently running.

Notifications can take several forms, and are platform-dependent:

- Alert: a pop-up text message
- Badge, Tile: a graphical representation that includes a short text or image
- Banner, Toast: a pop-up text message at the top of the device display that disappears after it has been read

- Audio alert

The IBM MobileFirst Platform Foundation unified push notification mechanism enables sending of mobile notifications to mobile devices. 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 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.

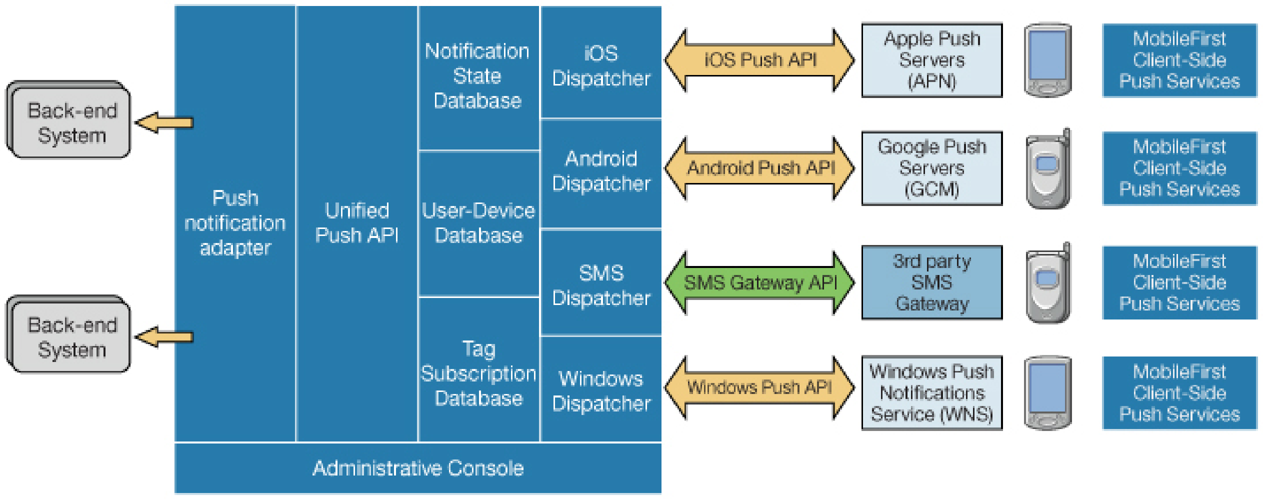

Back-end System
Back-end System
Push notification adapter
Unified Push API
Notification State Database
User-Device Database
Tag Subscription Database
iOS Dispatcher
Android Dispatcher
SMS Dispatcher
Windows Dispatcher
Administrative Console
iOS Push API
Android Push API
SMS Gateway API
Windows Push API
Apple Push Servers (APN)
Google Push Servers (GCM)
3rd party SMS Gateway
Windows Push Notifications Service (WNS)
MobileFirst Client-Side Push Services
MobileFirst Client-Side Push Services
MobileFirst Client-Side Push Services
MobileFirst Client-Side Push Services

Figure 7-18. Push notification mechanism

Push notification currently works for SMS, WNS, iOS and Android. iOS apps use the Apple Push Notification Service (APNs), Android apps use Google Cloud Messaging (GCM), and Windows apps use Windows Push Notification Services (WNS). For more information about setting up push notification for each platform, see “Setting up push notifications” on page 7-253.

Proxy settings

Use the proxy settings to set the optional proxy through which notifications are sent to APNs and GCM. You can set the proxy by using the **`push.apns.proxy.*`** and **`push.gcm.proxy.*`** configuration properties. For more information, see “List of JNDI properties for MobileFirst Server push service” on page 6-186.

Note: WNS does not have proxy support.

Architecture

Unlike other IBM MobileFirst Platform Foundation services, the push server requires outbound connections to Apple and Google servers using ports that are defined by these companies.

Push notification architecture

You can create an IBM MobileFirst Platform Foundation push notification architecture using the enterprise back-end calling method, in which an enterprise back end uses a IBM MobileFirst Platform Foundation adapter to deliver messages to a MobileFirst Server cluster.

This architecture relies on the enterprise back-end system to deliver messages to a MobileFirst Server cluster by calling push REST APIs.

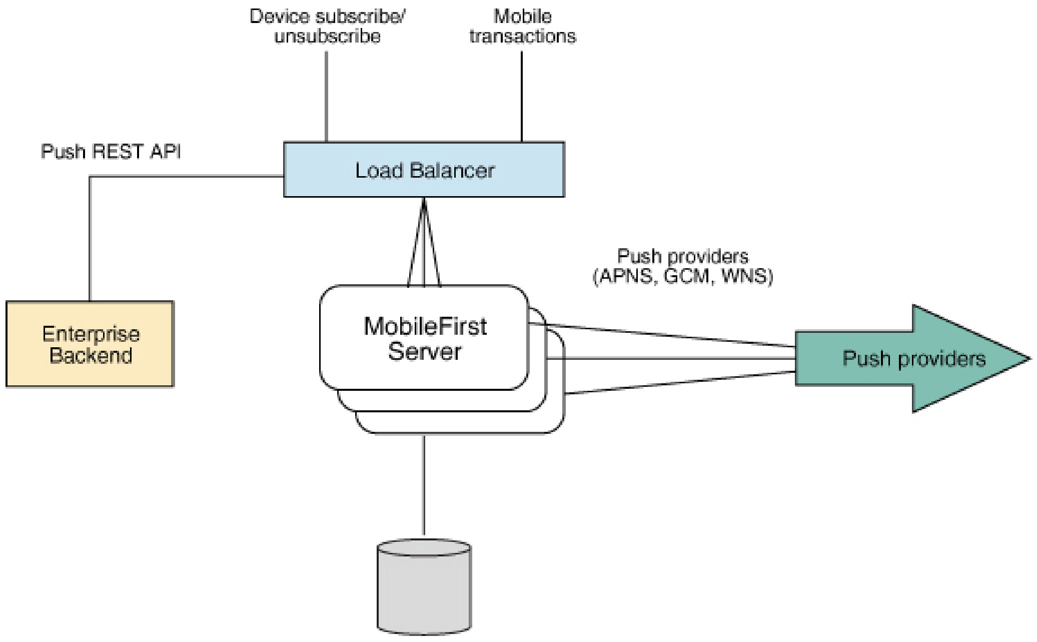


Figure 7-19. Enterprise back-end push notification architecture

With this architecture, 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 back-end initiates calls to the load balancer.

The advantage 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. The disadvantage of this method is 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 IBM MobileFirst Platform Foundation adapter procedure that is invoked call the back-end for a batch of messages rather than single messages.

Getting started with push notifications

Learn how to get started to add push notification support to your IBM MobileFirst Platform Foundation app.

Before you begin

- Set up your development environment. For more information, see "Setting up the development environment" on page 7-9.
- Create an application. For more information, see "Developing the client side of a MobileFirst application" on page 7-24.

About this task

To get started with push notification, go to the Development Center website and review the tutorials about push notification. However, if you need to learn how to migrate an existing IBM MobileFirst Platform Foundation to V8.0.0, read about the concept changes in "Migrating to push notifications from event source-based notifications" on page 5-54 and follow the instructions in "Migration scenarios" on page 5-56.

Procedure

1. Go to the Notifications tutorial page on the Development Center website.
2. Review the tutorials for your preferred development platform.

Security for push notification clients

Every client interacting with push must provide a valid access token with the required scopes.

For mobile client applications, IBM MobileFirst Platform Foundation SDK orchestrates the OAuth flow so that the mobile client application obtains a valid access token with the required scope.

The back-end server applications must register as confidential clients and must also implement the OAuth flow with the IBM MobileFirst Platform Foundation authorization server to obtain a valid access token with the required scopes.

For information on push scopes and the semantics that server applications can use as appropriate when obtaining a token, see Table 7-22. For information on configuring a confidential client, see "Confidential clients" on page 7-279.

Table 7-22. Push scopes and semantics

Scope	Meaning
apps.read	Permission to read application resource.
apps.write	Permission to create, update, delete application resource.
gcmConf.read	Permission to read GCM configuration settings (API Key and SenderId).
gcmConf.write	Permission to update, delete GCM configuration settings.
apnsConf.read	Permission to read APNs configuration settings.
apnsConf.write	Permission to update, delete APNs configuration settings.
devices.read	Permission to read device.
devices.write	Permission to create, update delete device.
subscriptions.read	Permission to read subscriptions.

Table 7-22. Push scopes and semantics (continued)

Scope	Meaning
subscriptions.write	Permission to create, update, delete subscriptions.
messages.write	Permission to send push notifications.
webhooks.write	Permission to read event-notifications.
webhooks.read	Permission to read event-notifications.
smsConf.read	Permission to read SMS configuration settings.
smsConf.write	Permission to update, delete SMS configuration settings.
wnsConf.read	Permission to read WNS configuration settings.
wnsConf.write	Permission to update, delete WNS configuration settings.

Related links

"Obtaining tokens"
Every client interacting with push must provide a valid access token with the required scopes for making Push REST API calls. A simple example on how to obtain the token and use the push REST API is shown.

Obtaining tokens

Every client interacting with push must provide a valid access token with the required scopes for making Push REST API calls. A simple example on how to obtain the token and use the push REST API is shown.

About this task

For mobile client applications, IBM MobileFirst Platform Foundation SDK orchestrates the OAuth flow so that the mobile push client application obtains a valid access token with the required scope.

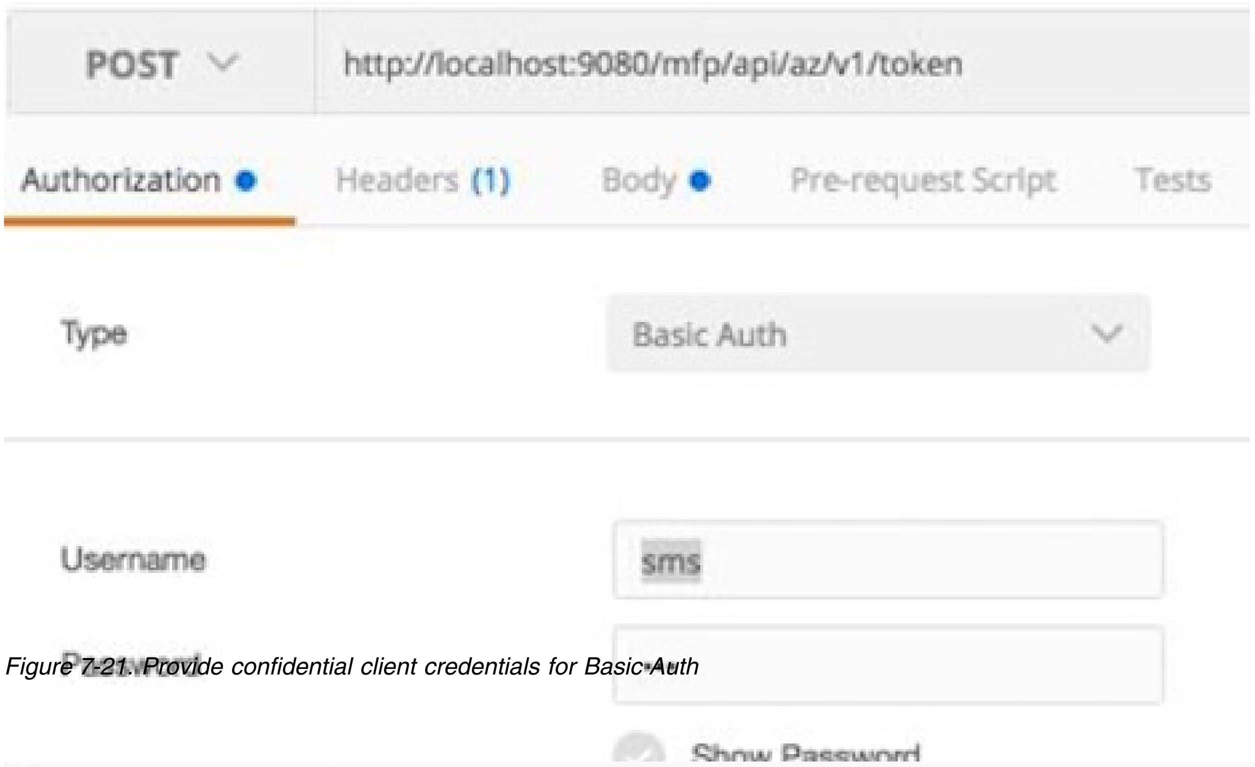
Procedure

1. You will have to POST a request to the URL http(s)://<host>:<port>/mfp/api/az/v1/token to get an access token.
2. Before you can POST the request, you will need to set the scope parameters in the **Body**.

   For information on push scopes and the semantics that server applications can use as appropriate when obtaining a token, see Table 7-22 on page 7-251.

Figure 7-20. Setting the scope

3. Now set the **Authorization** header by providing the confidential client credentials. For information on configuring a confidential client, see "Confidential clients" on page 7-279.


POST http://localhost:9080/mfp/api/az/v1/token

Authorization ● Headers (1) Body ● Pre-request Script Tests

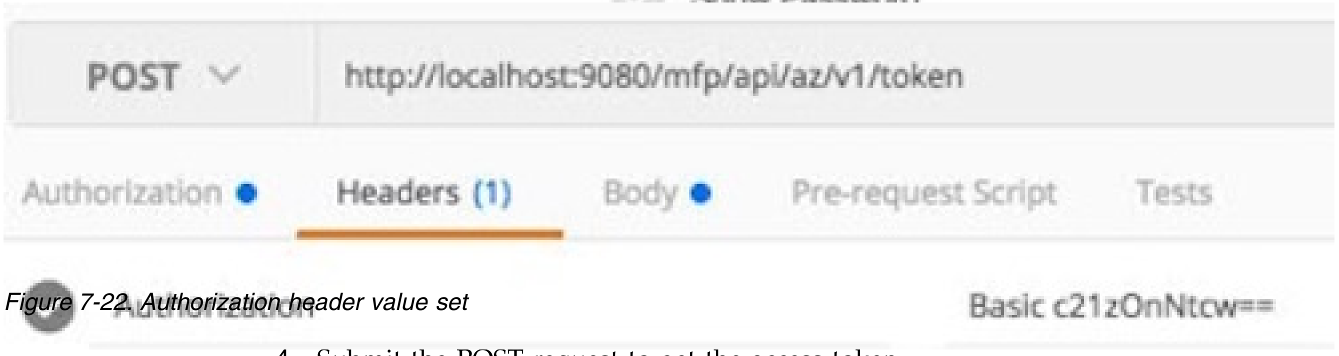
Type: Basic Auth

Username: sms

Password

Show Password

Figure 7-21. Provide confidential client credentials for Basic Auth

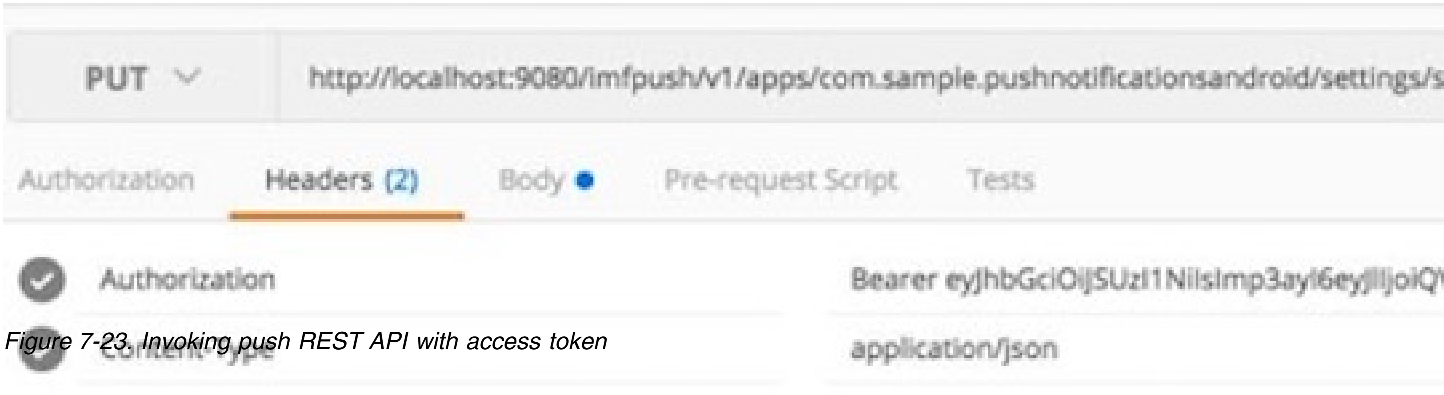

POST http://localhost:9080/mfp/api/az/v1/token

Authorization ● Headers (1) Body ● Pre-request Script Tests

Authorization: Basic c21zOnNtcw==

Figure 7-22. Authorization header value set

4. Submit the POST request to get the access token.


PUT http://localhost:9080/imfpush/v1/apps/com.sample.pushnotificationsandroid/settings/s

Authorization Headers (2) Body ● Pre-request Script Tests

Authorization: Bearer eyJhbGciOiJSUzI1NiIsImp3ayI6eyJlIjoiQ

Content-Type: application/json

Figure 7-23. Invoking push REST API with access token

Access token is obtained and is set as value of the **Authorization** header in the subsequent push REST API invocation.

Setting up push notifications

You can send push notifications to mobile devices via the MobileFirst Server. You can set up push notifications on Android and iOS.

About this task

The process for setting up push notifications varies significantly for each platform, and for Android and iOS you must refer to documentation for those products. For more information about the processes for each platform, see the following tasks:

Setting up push notifications for Android

To set up push notifications for Android devices and to enable push for Cordova applications for Android, you must use the Google Cloud Messaging (GCM) service. In order to use GCM, you need a valid Google account.

Before you begin

Before you set up push notifications for Android in IBM MobileFirst Platform Foundation, you must have an existing Google API project in the Google Developers Console (http://code.google.com/apis/console). This project must have a server key credential defined.

To configure a new API project in the Google Developers Console, go to https://developers.google.com/mobile/add

For more information about the credentials required for GCM, review the GCM components and credentials table descriptions on the Google Cloud Messaging: Oveview page at Google Developers.

Procedure

1. Gather the following information about your Google API project from Google Developers Console (http://code.google.com/apis/console):

   Project number

   The project number is a globally unique numerical value created when you create your Google API project. Be careful not to use either the project name or project ID as the `senderID` value.

   You can find the project number in the Google Developers Console dashboard by expanding your project and recording the value under **Project number**.

   Server key

   Make sure that the server key is not restricted to any specific URL. For more information about how to create the key, see API keys.

   You can get your server API from Credentials page in Google Developers Console by selecting **API Manager** > **Credentials**.
2. If your organization has a firewall that restricts the traffic to or from the Internet, you must do the following steps:
   a. Configure the firewall to allow connectivity with GCM in order for your GCM client apps to receive messages. The ports to open are 5228, 5229, and 5230. GCM typically uses only 5228, but it sometimes uses 5229 and 5230. GCM does not provide specific IP, so you must allow your firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169. For more information, see Implementing an HTTP Connection Server.
   b. Ensure that your firewall accepts outgoing connections from IBM MobileFirst Platform Foundation push notification service to `android.googleapis.com` on port 443.
3. You can set the certificates using any of the following methods:

- Using the IBM MobileFirst Platform Operations Console. See "Configuring push notification settings" on page 7-259.
- Using "Push GCM Settings (PUT)" on page 8-228 API.
- Using the "REST API for the MobileFirst Server administration service" on page 8-7.

4. To set up Google Play Services in your Android project, see https://developers.google.com/android/guides/setup.

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 APNs certificate for your application.

Before you begin

Ensure that the following servers are accessible from IBM MobileFirst Platform Foundation:

- 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. You can set the certificates using any of the following methods:
   - Using the IBM MobileFirst Platform Operations Console. See "Configuring push notification settings" on page 7-259.
   - Using "Push APNS settings (PUT)" on page 8-225 API.
   - Using the "REST API for the MobileFirst Server administration service" on page 8-7.
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.

Results

Your push notification setup is now complete.

Setting up push notifications for Windows

To set up push notifications for Windows devices, you must use the Windows Push Notification Services (WNS). In order to use WNS, you must have a valid Windows Store account.

Procedure

1. Create a Windows Store account at Windows Dev Center Dashboard.
2. Register your app with Windows Dev Center Dashboard and obtain the following credentials for your app.
   - Name
   - Publisher
   - Package Security Identifier (SID)
   - Client Secret
3. You can set the certificates using any of the following methods:
   - Using the IBM MobileFirst Platform Operations Console. See "Configuring push notification settings" on page 7-259.
   - Using "Push WNS Settings (PUT)" on page 8-232 API.
   - Using the "REST API for the MobileFirst Server administration service" on page 8-7.

   You can also associate your Windows project with the application in the Windows Store by right-clicking on the project and selecting **Store** > **Associate App** with the Store.

Results

Your push notifications setup is now complete.

Broadcast notifications

Broadcast notifications are notification messages that are targeted to all the devices that have the IBM MobileFirst Platform Foundation application installed and configured for push notifications.

Broadcast notifications are enabled by default with any IBM MobileFirst Platform Foundation application that is enabled for push notification. For more information about configuring your application for push notifications, see "Setting up push notifications" on page 7-253.

Any IBM MobileFirst Platform Foundation 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 native app, use `unsubscribetag` method of MFPPush class, with the tag name `Push.ALL`.

If you want to disable broadcast notification for hybrid app, use the `unsubscribetag` method of MFPPush class. To disable broadcast notification for Android native, use the `unsubscribetag` method of "Java client-side push API for Android apps" on page 8-6.

For more information about sending broadcast notification, see "Broadcast notification" on page 7-258 section in "Sending push notifications" on page 7-258.

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 you to define tags and send or 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 7-258 section in "Sending push notifications" on page 7-258.

Setting up tag-based notifications

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.

About this task

In a scenario where there are multiple users of a device, subscriptions should be implemented in mobile applications based on user log-in 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, subscribe the application to the tags that you have defined. To set up tag subscriptions, use the methods of the MFPPush class. For native application, use the methods of MFPPush class.

Results

While the tag subscription exists, IBM MobileFirst Platform Foundation can produce push notifications for the subscribed tag.

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 7-258 section in "Sending push notifications" on page 7-258.

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 IBM MobileFirst Platform Foundation application is enabled for push notifications. For more information about configuring your application for push notifications, see "Setting up push notifications" on page 7-253.

For more information about sending Unicast notification, see “Unicast notification” section in “Sending push notifications.”

Note: Unicast notification does not contain any tag in its payload.

Sending push notifications

When you have set up push notification, as tag-based or broadcast-enabled, you can send push notifications from the server.

You can send push notifications using the methods:
- By using MobileFirst Operations Console, two types of notifications can be sent: `tag` and `broadcast`. For more information, see “Sending push notification with the MobileFirst Operations Console” on page 7-259.
- By using “Push Message (POST)” on page 8-236 REST API. All forms of notifications can be sent: `tag`, `broadcast`, and `authenticated`.
- By using “REST API for the MobileFirst Server administration service” on page 8-7. All forms of notifications can be sent: `tag`, `broadcast`, and `authenticated`.

You can send the following notifications using the push notification service:

Broadcast notification

You must set up broadcast notifications to send a notification for the required applications.

For more information, see “Broadcast notifications” on page 7-256.

Tag-based notification

You must set up tag subscriptions to send a notification for the required applications.

For more information, see “Setting up tag-based notifications” on page 7-257.

Unicast notification

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: Unicast notification does not contain any tag in its payload. The notification message can target multiple devices or users by specifying multiple **`deviceIDs`** or **`userIDs`** respectively, in the target block of POST message API.

Platform or environment-based notification

You can send a platform or environment-based notification in the following way:
- Specify the platform or platforms as an array in the **`target.platform`** object. The supported platforms are as follows:
  - A (Apple)
  - G (Google)
  - W (Windows)

Sending push notification with the MobileFirst Operations Console

The IBM MobileFirst Platform Operations Console provides a GUI for an administrator to work with push notification to update credentials, setup tags and send notifications.

About this task

For IBM MobileFirst Platform Foundation configuration with push notification service enabled, every application created would automatically be registered with push notifications. The application contains configuration information; such as the Apple Push Notification Service (APNs) and Google Cloud Message (GCM) configuration details. This information is required by the push notification service to send messages.

The following topics guides you on configuring your push notification settings, creating tags and sending notifications. You can also go through the steps on how to configure a confidential client and enable scope mapping elements to security checks.

Configuring push notification settings:

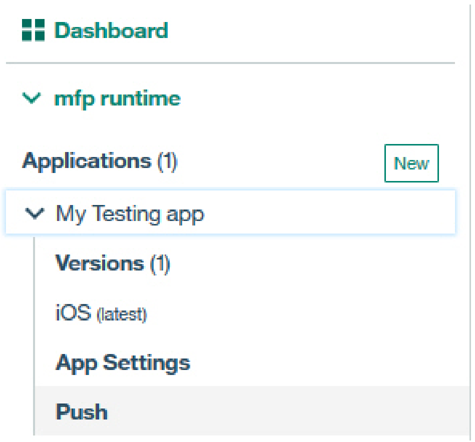
You must configure push notification for every app that is created.

About this task

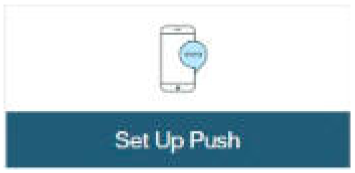
Complete the steps to configure the push notification settings:

Procedure

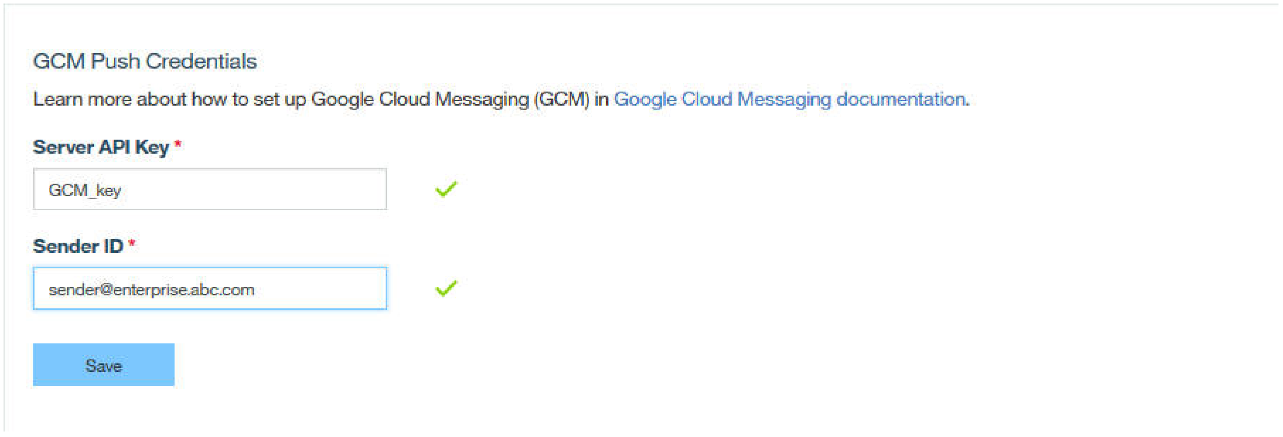
1. To configure the push notification settings for your application, use one of the following methods:
   - Expand **mfp runtime** > **Applications** > **application name** > **Push**.



   - Click the **Set Up Push** icon in the **Next Steps** section of the main page for your app.



2. In the Push Notifications Settings panel, proceed as follows:
   - Update the Server API Key and Sender ID fields. Click **Save**.



GCM Push Credentials

Learn more about how to set up Google Cloud Messaging (GCM) in Google Cloud Messaging documentation.

Server API Key *

GCM_key ✓

Sender ID *

sender@enterprise.abc.com ✓

Figure 7-24. Providing the GCM Push Credentials

Creating tags for push notification:

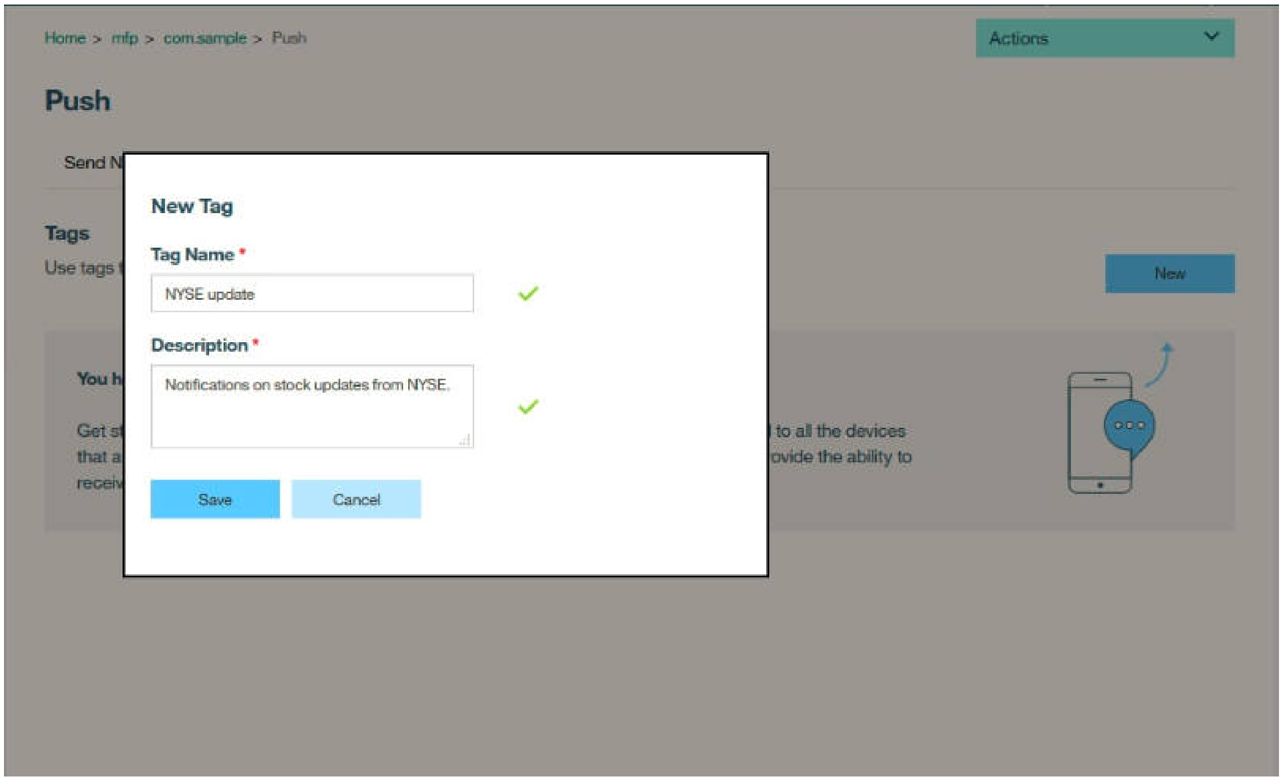
By creating tags, you can enable push notifications to be sent to subscribers.

About this task

You can send push notification to users who have chosen to subscribe to tags. To create a tag, complete the steps:

Procedure

1. In the Tags pane, click **New** to create a new tag.
2. In the New Tag window, provide a suitable name and description. Click **Save**.



New Tag

Tag Name *

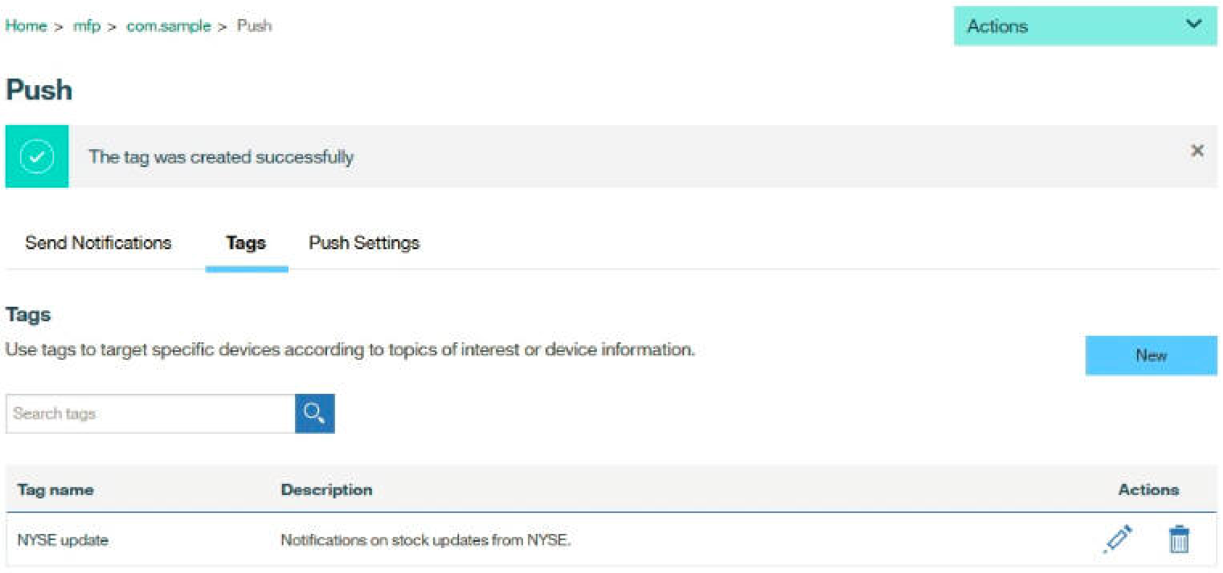
NYSE update

Description *

Notifications on stock updates from NYSE.

Figure 7-25. Providing a tag name and description

The new tag is generated.



The tag was created successfully

Tags

Use tags to target specific devices according to topics of interest or device information.

Tag name	Description	Actions
NYSE update	Notifications on stock updates from NYSE.	

Figure 7-26. Tags created successfully

Sending push notifications to subscribers:

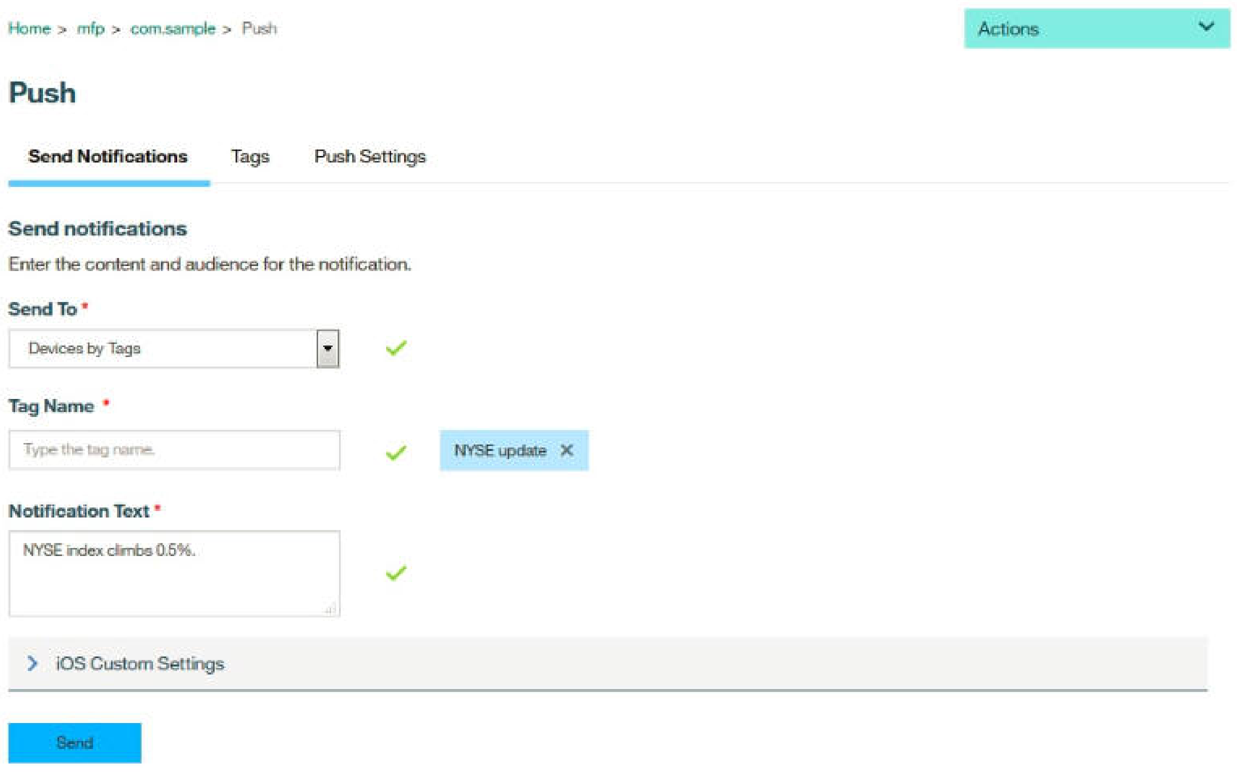
After configuring push notification settings and creating tags, you can choose to send either tag-based notifications or broadcast notifications to subscribers.

About this task

Complete the steps to send push notifications to subscribers:

Procedure

1. Click the **Send Notifications** tab to choose and select the notifications that needs to be sent.
2. Choose to send notifications based on any of the following criteria:
   - **All**: Broadcast message. Sends notifications to all subscribers and devices.
   - **Devices by Tags**: Tag-based notification. Tags represent topics of interest to the user. This option sends notifications to subscribers who might have subscribed to a particular tag. The **Tag Name** and **Notification Text** fields are mandatory.
   - **Single device**: Broadcast message. To send notifications only to a specified device. The **Device ID** and **Notification Text** fields are mandatory.
   - **iOS devices**: Broadcast message. To send notifications to all iOS devices.



Send notifications

Enter the content and audience for the notification.

Send To *: Devices by Tags

Tag Name *: Type the tag name. NYSE update ✕

Notification Text *: NYSE index climbs 0.5%.

Figure 7-27. Send Notifications pane

3. Click **Send** to send push notification to subscribers.

Defining scope mapping elements to security checks:

IBM MobileFirst Platform Foundation allows mapping custom scope elements to security checks with which you can define application-specific security logic for accessing protected resources.

About this task

Your app would need scope element **`push.mobileclient`** to be defined, for the client app to access push server. No security checks will be enforced if you do not map the scope to security checks.

Complete the steps to define the scope elements:

For information on push scopes, see Table 7-22 on page 7-251.

Procedure

1. In the MobileFirst Operations Console, navigate to ***<Your application>*** > **Security** > **Scope Elements Mapping** > **New**.
2. In the Add New Scope-Element Mapping window, provide the scope element as **`push.mobileclient`** and click **Add**.

   Note: The **`push.mobileclient`** is a predefined scope.

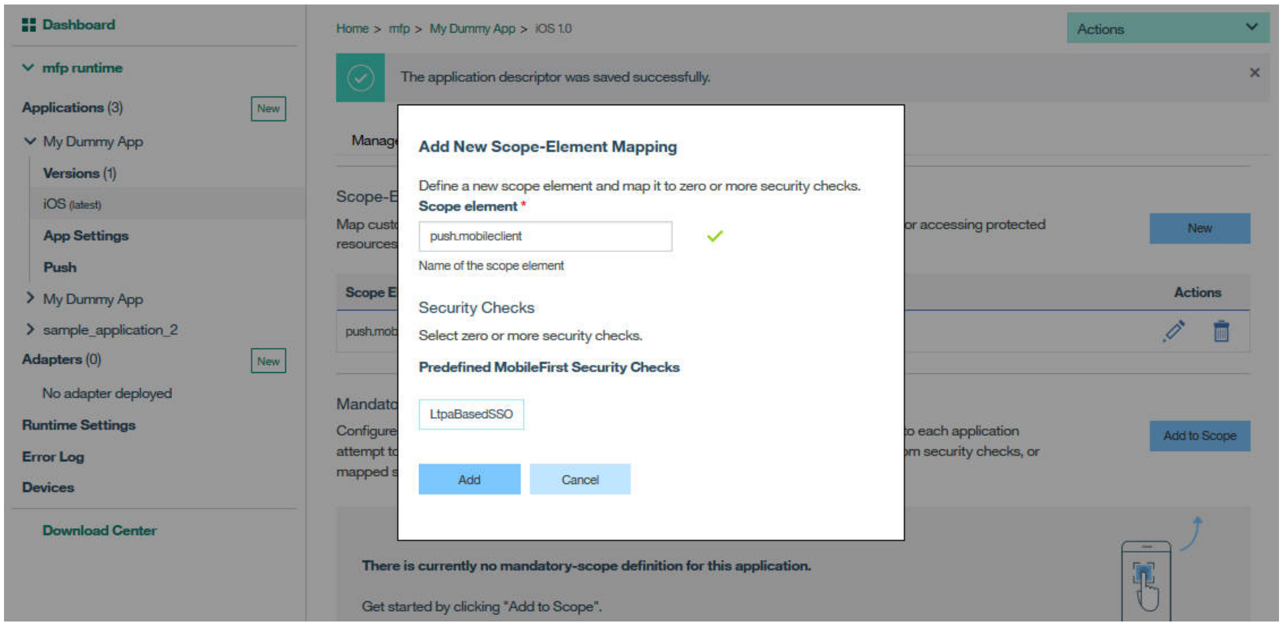

Dashboard
mfp runtime
Applications (3)
New
My Dummy App
Versions (1)
iOS (latest)
App Settings
Push
My Dummy App
sample_application_2
Adapters (0)
New
No adapter deployed
Runtime Settings
Error Log
Devices
Download Center
Home > mfp > My Dummy App > iOS 1.0
Actions
The application descriptor was saved successfully.
Add New Scope-Element Mapping
Define a new scope element and map it to zero or more security checks.
Scope element *
push.mobileclient
Name of the scope element
Security Checks
Select zero or more security checks.
Predefined MobileFirst Security Checks
LtpaBasedSSO
Add
Cancel
New
Actions
Add to Scope
There is currently no mandatory-scope definition for this application.
Get started by clicking "Add to Scope".

Figure 7-28. Adding new scope element

Configuring a confidential client:

IBM MobileFirst Platform Foundation V8.0.0 allows you to connect a confidential (or non-mobile) client to mobile services in a secure way.

About this task

You can provide confidential applications with a back-end service access to the push notification service. Ensure that you have gone through the steps to register the confidential client.

Procedure

1. Register the confidential client.

   For more information, see "Registering confidential clients" on page 7-279.
2. Be sure to become familiar with push scopes and their semantics, so that server applications use them as appropriate when they obtain a token.

   For more information about push scopes, see "Security for push notification clients" on page 7-251.

Sending SMS notifications

You can send short message service (SMS) messages, commonly known as text messages, to user devices. To be able to receive SMS notifications, user must register to the notification by using their phone number.

The SMS notification framework extends the push notification framework. SMS notification capability is supported for Apple, Google, and Windows Phone 8 devices that support SMS functions.

Setting up SMS notification

You can send short message service (SMS) messages, commonly known as text messages, to user devices. Learn about the configuration that is required to send and receive SMS notifications.

About this task

The procedure to set up push server to send SMS notifications is described.

Procedure

1. Set up the SMS notification infrastructure by updating the SMS settings.

   Refer to Push SMS Settings (PUT) API for details.
2. To receive SMS notifications, register the device with the phone number by following one of the approaches:
   - From the app, provide the **phone number** options parameter of the `registerDevice` method in the MFPPush.
   - Use Push Device Registration (POST) API with the phone number in the payload.
3. To send SMS notifications, use the Push Message (POST) API.

   Set the **notificationType** parameter in the payload to a value of `2` or `3`.

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 for the MobileFirst Server push service" on page 8-197 to access Push functions from a REST API endpoint.

Troubleshooting push notification problems

Find information to help resolve push notification issues that you might encounter.

iOS push notification

Problem

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 key decryption error:
(java.security.InvalidKeyException: Illegal key size)

at com.notnoop.apns.internal.Utilities.newSSLContext(Utilities.java:88)

at com.ibm.pushworks.server.notification.apns.ApplicationConnection.createBuilderWithCertificate(ApplicationConnection.java:180)
at com.ibm.pushworks.server.notification.apns.ApplicationConnection.<init>(ApplicationConnection.java:59)

...
```

Actions to take

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 newer versions (Version 1.4.2 and higher).
   c. Click **Continue** and finish the download process.

      The `.zip` archive contains the following 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 found 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

The MobileFirst security framework implements the security capabilities of IBM MobileFirst Platform Foundation, which are used to secure your mobile applications and protect your resources.

Overview of the MobileFirst security framework

Learn about the security framework of IBM MobileFirst Platform Foundation (the security framework), its building blocks, and the related authorization flows for protecting your resources and securing your applications.

The security framework is based on the OAuth 2.0 protocol, as defined in the OAuth Specification. According to this protocol, a resource can be protected by a scope that defines the required permissions for accessing the resource. To access a protected resource, the client must provide a matching access token, which encapsulates the scope of the authorization that is granted to the client.

The OAuth protocol separates the roles of the authorization server and the resource server on which the resource is hosted. The authorization server manages the client authorization and token generation. The resource server uses the authorization server to validate the access token that is provided by the client, and ensure that it matches the protecting scope of the requested resource.

The security framework is built around an authorization server that implements the OAuth protocol, and exposes the OAuth endpoints with which the client interacts to obtain access tokens. The framework provides the building blocks for implementing a custom authorization logic on top of the authorization server and the underlying OAuth protocol. By default, MobileFirst Server functions also as the authorization server. However, you can configure an IBM WebSphere DataPower appliance to act as the authorization server and interact with MobileFirst Server.

OAuth scopes, security checks, and challenge handlers

In the OAuth model, a resource is protected by a scope, which is string of zero or more space-separated scope elements. Each scope element represents a logical authorization permission. The MobileFirst security framework maps scope elements into security checks, which implement the actual authorization logic.

A security check is a server-side entity that implements the security logic for protecting server-side application resources. A simple example of a security check is a user-login security check that receives the credentials of a user, and verifies the credentials against a user registry. Another example is the predefined MobileFirst application-authenticity security check, which validates the authenticity of the mobile application and thus protects against unlawful attempts to access the application's resources. Server-side developers can write security checks that implement their required authorization logic. The framework also contains predefined security checks, for example for validating the authenticity of a mobile application.

A security check typically issues security challenges that require the client to respond in a specific way to pass the check. This handshake occurs as part of the OAuth access-token-acquisition flow. The client uses challenge handlers to handle challenges from security checks. A challenge handler is a client-side entity that implements the client-side security logic and the related user interaction.

For each security check that issues a challenge, a matching client challenge handler must be registered in the application code. The MobileFirst client API includes preregistered challenge handlers for the predefined security checks. To support a custom security check, the client-side developer must implement and register a challenge handler for that security check in the application code.

Application developers protect access to their resources by defining the required scope for each protected resource, and implementing the related security checks and challenge handlers. The server-side security framework and the client-side API handle the OAuth message exchange and the interaction with the authorization server transparently, allowing developers to focus only on the authorization logic.

End-to-end authorization flow

Following is an outline of the end-to-end flow for authorizing client access to a protected resource. The flow consists of two main stages:

1. The client obtains an access token for the protected resources, after passing the required security checks (as defined in the resource's protecting scope). See Obtaining an access token.
2. The client uses the access token to access the protected resource. Before granting the client access to the resource, the access token is validated. See Accessing a protected resource by using an access token.

Obtaining an access token

The client obtains an access token from the authorization server by following these steps, as illustrated in Figure 1:

1. Registration - the client registers itself with MobileFirst Server. As part of the registration, the client provides a public key that will be used for authenticating its identity (see Step 4). This phase occurs once in the lifetime of a mobile application instance. If the application-authenticity security check is enabled for a mobile client application, the authenticity of the application is validated during its registration (see "Application-authenticity security check" on page 7-282).
2. Access-token request - the client requests an access token with a certain scope. The requested scope should map to the same security checks as the scope of the protected resource that the client wants to access, and can optionally also map to additional security checks.
   If the client does not have prior knowledge about the scope of the protected resource, it can first request an access token with an empty scope, and try to access the resource with the obtained token. The client will receive a response with a 403 (Forbidden) error and the required scope of the requested resource.
3. Authorization - MobileFirst Server runs the security checks to which the scope of the client's request is mapped. The authorization server either grants or rejects the client's request based on the results of these checks. If a mandatory application scope is defined, the security checks of this scope are run in addition to the checks of the requested scope.
4. Token generation - after successful authorization, the client is redirected to the authorization server's token endpoint, where it is authenticated by using the public key that was provided as part of the client's registration (see Step 1). Upon successful authentication, the authorization server issues the client a digitally signed access token that encapsulates the client's ID, the requested scope, and the token's expiration time.

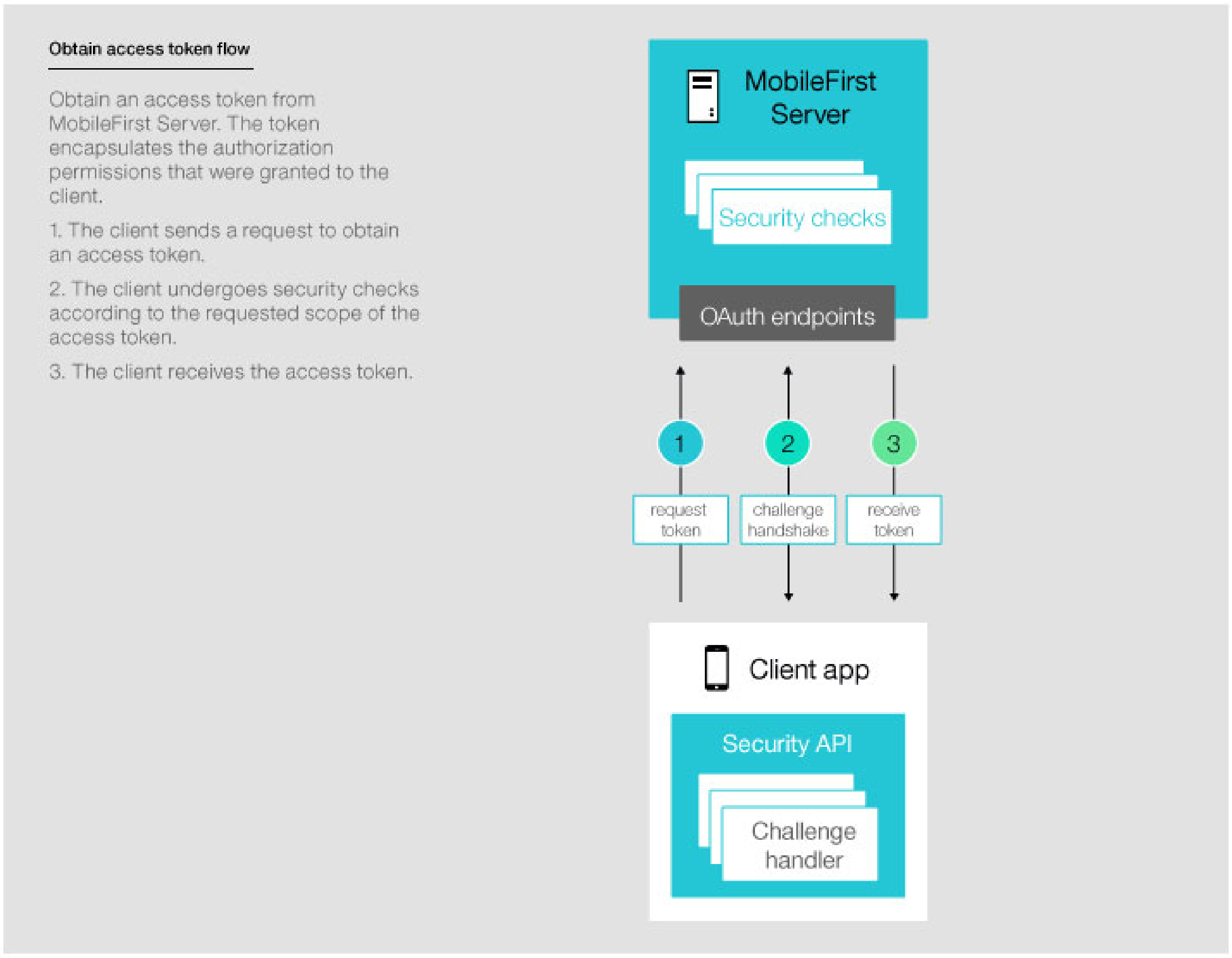

Obtain access token flow
Obtain an access token from MobileFirst Server. The token encapsulates the authorization permissions that were granted to the client.
1. The client sends a request to obtain an access token.
2. The client undergoes security checks according to the requested scope of the access token.
3. The client receives the access token.
MobileFirst Server
Security checks
OAuth endpoints
1
2
3
request token
challenge handshake
receive token
Client app
Security API
Challenge handler

Figure 7-29. Obtaining an access token

Note: The "MobileFirst Server" box in Figure 1 embodies both the functions that are provided specifically by MobileFirst Server, and the functions that are provided by the authorization server. The authorization server can be either MobileFirst Server (default) or WebSphere DataPower.

Accessing a protected resource by using an access token

After obtaining an access token, the client attaches the obtained token to subsequent requests to access protected resources. The resource server uses the authorization server's introspection endpoint to validate the token. The validation includes using the token's digital signature to verify the client's identity, verifying that the scope matches the authorized requested scope, and ensuring that the token has not expired. When the token is validated, the client is granted access to the resource. The protected resource can be hosted on an instance of MobileFirst Server (see Figure 2) or on an external server (see Figure 3).

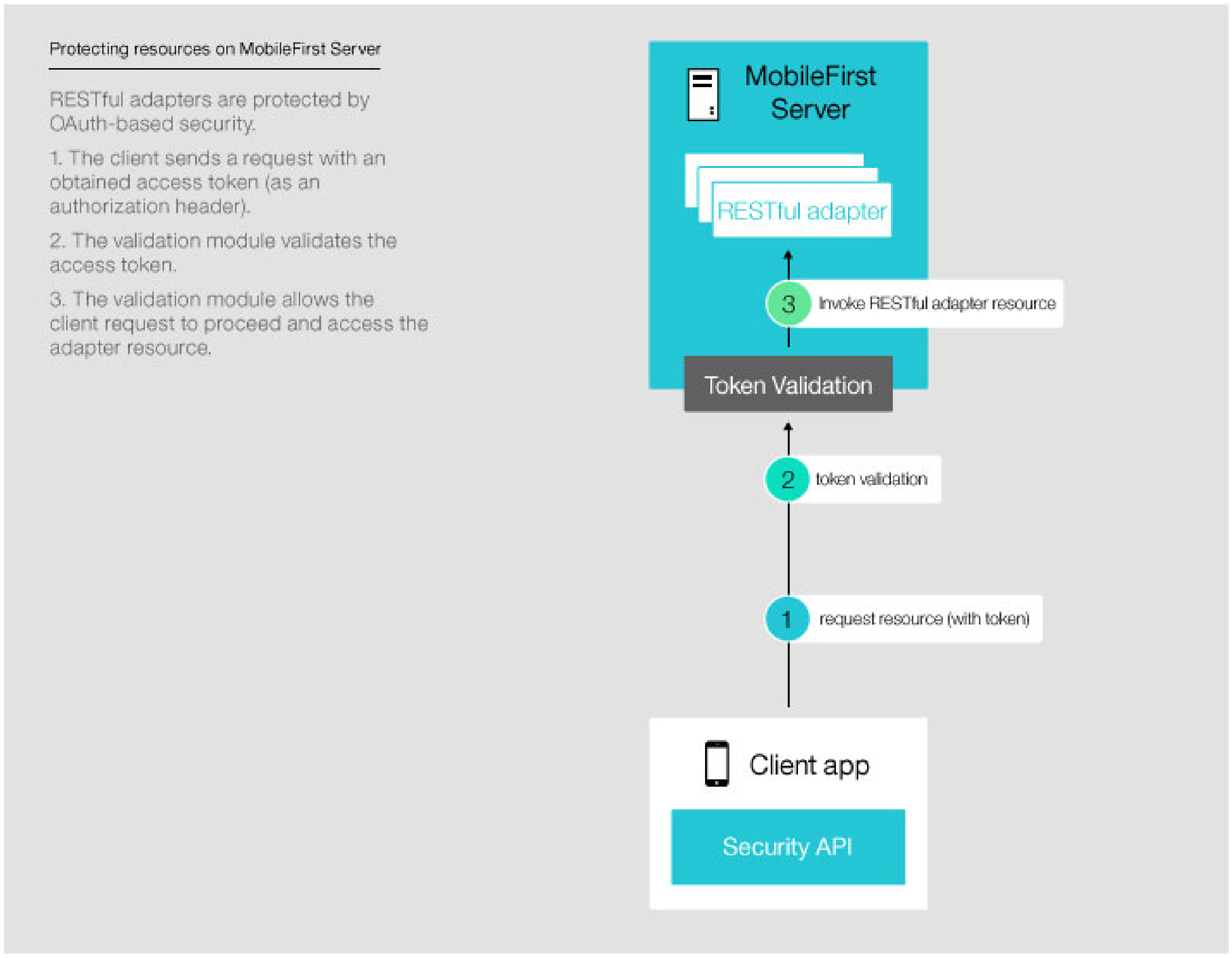

Protecting resources on MobileFirst Server
RESTful adapters are protected by OAuth-based security.
1. The client sends a request with an obtained access token (as an authorization header).
2. The validation module validates the access token.
3. The validation module allows the client request to proceed and access the adapter resource.
MobileFirst Server
RESTful adapter
3 Invoke RESTful adapter resource
Token Validation
2 token validation
1 request resource (with token)
Client app
Security API

Figure 7-30. Protecting a resource on MobileFirst Server

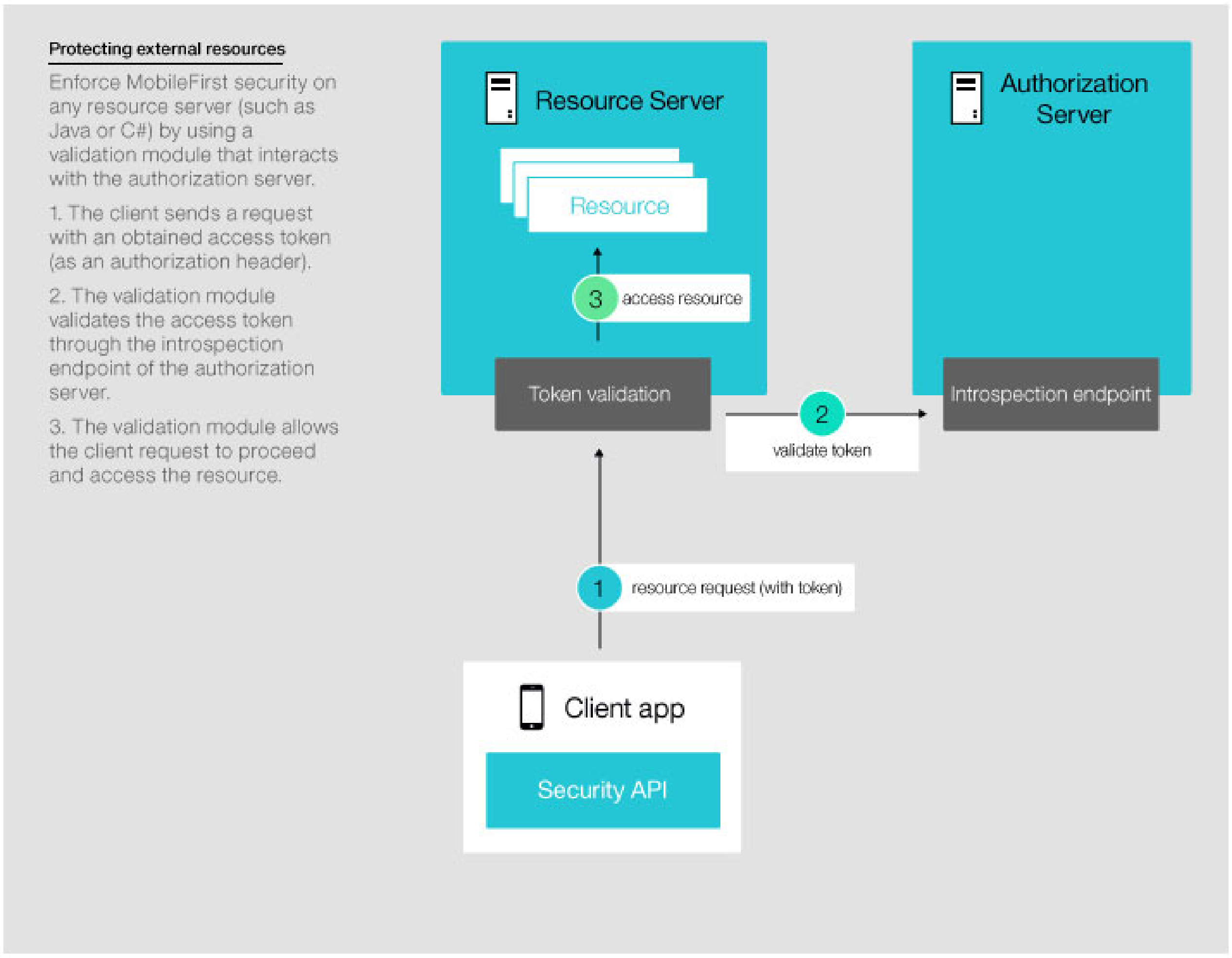

Protecting external resources
Enforce MobileFirst security on any resource server (such as Java or C#) by using a validation module that interacts with the authorization server.
1. The client sends a request with an obtained access token (as an authorization header).
2. The validation module validates the access token through the introspection endpoint of the authorization server.
3. The validation module allows the client request to proceed and access the resource.
Resource Server
Resource
3 access resource
Token validation
2 validate token
Authorization Server
Introspection endpoint
1 resource request (with token)
Client app
Security API

Figure 7-31. Protecting a resource on an external server

Note: The "MobileFirst Server" box in Figure 2 and Figure 3 embodies both the functions that are provided specifically by MobileFirst Server, and the functions that are provided by the authorization server. The authorization server can be either MobileFirst Server (default) or WebSphere DataPower.

Related information

For more detailed information and specific development guidelines, see the following related topics:

- “Access tokens” on page 7-303
- “Client security APIs” on page 7-305
- “Configuring a mandatory application scope” on page 7-276
- “Configuring IBM WebSphere DataPower as the OAuth authorization server” on page 7-314
- “Endpoints of the MobileFirst Server production server” on page 6-164
- “Mapping scope elements” on page 7-277
- “OAuth resource protection” on page 7-271
- “Security checks” on page 7-281

OAuth resource protection

Learn how to configure and customize OAuth protection for your resources.

Protected resources

In the OAuth model, a protected resource is a resource that requires an access token. You can use the MobileFirst security framework to protect both resources that are hosted on an instance of MobileFirst Server, and resources on an external server. You protect a resource by assigning it a scope that defines the required permissions for acquiring an access token for the resource. See "Overview of the MobileFirst security framework" on page 7-265. Mobile-application access to protected resources is restricted also by the mandatory application scope.

MobileFirst adapter resources are protected by default, meaning that an access token is required to access such resources even when no scope is explicitly assigned to the resource. You can disable the default resource protection.

The resource scope can contain custom scope elements that are mapped to security checks at the application level.

Note: An empty scope is also a valid scope, and requires an access token.

Unprotected resources

An unprotected resource is a resource that does not require an access token. The MobileFirst security framework does not manage access to unprotected resources, and does not validate or check the identity of clients that access these resources. Therefore, features such as Direct Update, blocking device access, or remotely disabling an application, are not supported for unprotected resources. See "Updating Cordova client apps directly" on page 7-235 and "Mobile-application management" on page 10-15.

Configuring resource protection

- To configure protection of adapter resources that are hosted on MobileFirst Server, see Configure adapter resource protection.
  - Java API for RESTful Web Services (JAX-RS) adapter resources
  - JavaScript adapter resources
- To configure protection of resources that are hosted on an external server (external resources), see Protect resources on external servers.
  - Protect resources on any Java server. See "MobileFirst Java Token Validator" on page 7-274.
  - Protect resources on WebSphere Application Server Java servers (Full or Liberty profile). See "MobileFirst OAuth Trust Association Interceptor (TAI) for protecting resources on WebSphere Java servers" on page 7-275.
  - Protect resources on Node.js servers. See "MobileFirst Node.js resource protection" on page 7-275.
- To define a mandatory application scope, which is applied to any request by the application to access a protected resource, see "Configuring a mandatory application scope" on page 7-276.
- To map custom scope elements to security checks, see "Mapping scope elements" on page 7-277.

Configuring adapter resource protection

Learn how to configure MobileFirst OAuth protection for your adapter resources.

About this task

Configure the authorization logic for protecting your adapter resources by assigning custom scopes to your resources, or by disabling the default protection of the MobileFirst security framework (see "OAuth resource protection" on page 7-271).

- Configure protection of Java API for RESTful Web Services (JAX-RS) resources
  - Configure a resource scope
  - Disable resource protection
- Configure protection of JavaScript resources
  - Configure a resource scope
  - Disable resource protection

Procedure

Configure the protection of your adapter resources by following the outlined procedure for your target development environment:

- **Configure protection of Java API for RESTful Web Services (JAX-RS) resources**

  In Java, you configure resource protection by using the @OAuthSecurity annotation type, which is declared in the MobileFirst com.ibm.mfp.adapter.api package.
  This annotation can be applied either to a specific resource method or to an entire resource class. Method-level annotations override class-level annotations. The annotation can be used either to set the resource's protecting scope, or to disable resource protection and define an unprotected resource.

  Configure a resource scope

  To assign a protecting scope to a JAX-RS resource or resource class, add the @OAuthSecurity annotation to the resource or class declaration, and set the scope element of the annotation to your preferred scope:

  ```
  @OAuthSecurity(scope = "[scopeElement1 scopeElement2 ...]")
  ```

  Set the scope to a space-separated list of zero or more scope elements (see OAuth scopes). The default value of the annotation's scope element is an empty string.
  When the enabled element of the @OAuthSecurity annotation is set to `false`, the scope element is ignored. See Disable resource protection.

  Note: A class scope applies to all of the resources in the class, except for resources that have their own @OAuthSecurity annotation.

  Examples

  - The following code protects an helloUser method with a scope that contains `UserAuthentication` and `Pincode` scope elements:

    ```
    @GET
    @Path("/{username}")
    @OAuthSecurity(scope = "UserAuthentication Pincode")
    public String helloUser(@PathParam("username") String name){
        ...
    }
    ```

  - The following code protects a WebSphereResources class with the predefined LtpaBasedSSO security check:

```
@Path("/users")
@OAuthSecurity(scope = "LtpaBasedSSO")
public class WebSphereResources {
    ...
}
```

Disable resource protection

To entirely disable OAuth protection of your resource or resource class, add the @OAuthSecurity annotation to the resource or class declaration, and set the value of the enabled element to `false`:

```
@OAuthSecurity(enabled = false)
```

The default value of the annotation's enabled element is `true`. When the enabled element is set to `false`, the scope element is ignored, and the resource or resource class is not protected. See "Unprotected resources" on page 7-271.

Note: When you assign a scope to a resource method that is contained in an unprotected class, the method is protected despite the class annotation, provided you do not also set the enabled element to `false` in the resource annotation.

Examples

– The following code disables resource protection for a helloUser method:

```
@GET
@Path("/{username}")
@OAuthSecurity(enabled = "false")
public String helloUser(@PathParam("username") String name){
    ...
}
```

– The following code disables resource protection for a MyUnprotectedResources class.

```
@Path("/users")
@OAuthSecurity(enabled = "false")
public class MyUnprotectedResources {
    ...
}
```

- **Configure protection of JavaScript resources**

  In JavaScript, you configure resource protection as part of the definition of the adapter resource procedure, by setting the relevant attribute values of the <procedure> element in the adapter-descriptor (`adapter.xml` file). See the documentation of this element, and its subelements and attributes, in Structure of JavaScript adapters. You can configure the procedure either to set the resource's protecting scope, or to disable resource protection and define an unprotected resource.

  Configure a resource scope

  To assign a protecting scope to a JavaScript resource procedure, set the `scope` attribute of the <procedure> element to your preferred scope, as a space-separated list of zero or more scope elements (see OAuth scopes):

```
<procedure name="procedureName" scope="[scopeElement1 scopeElement2 ...]">
```

  When the `secured` attribute of the <procedure> element is set to `false`, the `scope` attribute is ignored. See Disable resource protection.

Example
The following code protects a userName procedure with a scope that contains `UserAuthentication` and `Pincode` scope elements:

```
<procedure name="userName" scope="UserAuthentication Pincode">
```

Disable resource protection
To entirely disable OAuth protection of your resource procedure, set the `secured` attribute of the <procedure> element to `false`:

```
<procedure name="procedureName" secured="false">
```

When the `enabled` attribute is set to `false`, the `scope` attribute is ignored, and the resource is not protected. See "Unprotected resources" on page 7-271

Example
The following code disables resource protection for a userName procedure:

```
<procedure name="userName" secured="false">
```

What to do next

Rebuild your adapter and deploy it to an instance of MobileFirst Server to apply your configuration.
When working with IBM MobileFirst Platform Operations Console, remember to refresh the console browser page after you deploy the adapter.

Before moving to production, make sure that the security checks that are contained in your configured scopes are implemented and available for your resources via an adapter that is deployed to the same MobileFirst Server instance as your resource adapter. See "Security-checks implementation" on page 7-289.

External resources protection

Learn how to use the MobileFirst security framework to protect resources that are stored on external servers (external resources).

To protect external resources, you add a resource filter with an access-token validation module to the external resource server. The token-validation module uses the introspection endpoint of the security framework's authorization server to validate MobileFirst access tokens before granting the OAuth client access to the resources. See "Overview of the MobileFirst security framework" on page 7-265, and specifically "Accessing a protected resource by using an access token" on page 7-268 and the illustration in Figure 3 (Protecting a resource on an external server).
You can use the MobileFirst REST API for the MobileFirst runtime to create your own access-token validation module for any external server. Alternatively, use one of the provided MobileFirst extensions for protecting external Java resources, as outlined in the following topics.

MobileFirst Java Token Validator:

Use the MobileFirst Java Token Validator access-token validation module to protect resources on any external Java server.

MobileFirst Java Token Validator is provided as a Java library (`mfp-java-token-validator-8.0.0.jar`). The library exposes an API that encapsulates and simplifies the interaction with the authorization server's introspection endpoint.

You can get a copy of the Java library Token Validator library by using any of the following methods:

- Download `mfp-java-token-validator` from the Maven repository.
- Get a copy of the library from the <*product_install_dir*>`/MobileFirstServer/external-server-libraries/` directory (where <*product_install_dir*> is the directory in which you installed IBM MobileFirst Platform Foundation).

For detailed information on how to install, configure, and use this validation module, see the Java Token Validator tutorial.

Note: To protect Java resources on WebSphere Application Server or WebSphere Application Server Liberty servers, you can use the MobileFirst OAuth TAI filter, which uses the Java Token Validator validation module. See "MobileFirst OAuth Trust Association Interceptor (TAI) for protecting resources on WebSphere Java servers."

MobileFirst OAuth Trust Association Interceptor (TAI) for protecting resources on WebSphere Java servers:

Use the MobileFirst OAuth Trust Association Interceptor (TAI) filter to protect Java resources that are hosted on a WebSphere Application Server with the Full or Liberty profile.

The MobileFirst OAuth TAI resource-server filter uses the MobileFirst Java Token Validator validation module (see "MobileFirst Java Token Validator" on page 7-274). The filter is provided as a Java library (`com.ibm.imf.oauth-8.0.0.jar`).

You can get a copy of the OAuth TAI filter by using any of the following methods:

- Download a copy of the library from theIBM MobileFirst Platform Operations Console: from the console **Dashboard**, select **Download Center**, and then select the **Tools** tab. In the **OAuth Security Java Extension** section, select **Download** and save the artifacts archive file to your preferred location.
- Get a copy of the library from the <*product_install_dir*>`/MobileFirstServer/external-server-libraries/` directory (where <*product_install_dir*> is the directory in which you installed IBM MobileFirst Platform Foundation).

For detailed information on how to install, configure, and use the MobileFirst OAuth TAI filter, see the Trust Association Interceptor tutorial.

MobileFirst Node.js resource protection:

Use the MobileFirst Node.js framework to protect Java resources (APIs) that are hosted on an external Node.js server.

The MobileFirst Node.js framework is provided as an npm module: `passport-mfp-token-validation`. This module provides a passport validation strategy and a verification function for validating access tokens that are issued by the MobileFirst security framework.

To install the `passport-mfp-token-validation` Node.js module, run the following command from the command line:

```
npm install passport-mfp-token-validation
```

For detailed information on how to install, configure, and use `passport-mfp-token-validation`, see the Node.js Validator tutorial.

Configuring a mandatory application scope

Configure a mandatory application scope to define application-specific authorization logic.

Before you begin

To use custom scope elements in your mandatory application scope, first map the required scope elements to security checks. See "Mapping scope elements" on page 7-277.

About this task

You can define a mandatory scope for your client application. When an application attempts to access a protected resource, the security framework maps the mandatory application scope to security checks. The framework runs these checks (if exist) in addition to the security checks of the requested resource scope. Follow the outlined procedure to define a mandatory application scope.

Note:

- As with any other security scope, the mandatory application scope is not applied when accessing an unprotected resource. See "Unprotected resources" on page 7-271.
- The access token that is granted for the resource scope does not contain the mandatory application scope. See "Structure of the MobileFirst access token" on page 7-304.

Procedure

Define the mandatory application scope by using one of the following alternative methods:

- **Using the IBM MobileFirst Platform Operations Console** (the console)
  1. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Security** tab.
  2. In the **Mandatory Application Scope** section, select **Create New**.
  3. In the Configure Mandatory Application Scope dialog window, select a scope element or security check from the items in the **Select elements and security checks** list, and select **Add**. The selection is from among custom scope elements that were mapped for your application, custom security checks defined in adapters that are deployed to the same MobileFirst Server instance as your application, and the predefined MobileFirst security checks.
     Repeat this step as needed to add more scope elements and security checks to the scope.

  To undo your configuration and eliminate the mandatory application scope, in the **Mandatory Application Scope** section of the console's application **Security** tab, delete all the scope elements that you previously added.
- **Editing the application-descriptor file**
  1. Create a local copy of the application-descriptor JSON file. See "Application configuration" on page 7-3.
  2. Edit your local copy to define a `mandatoryScope` property object, and set the property value to a scope string that contains a space-separated list of your selected scope elements:
     ```
     "mandatoryScope": "ScopeElement1 [ScopeElement2 ...]"
     ```

A scope element can be the name of a custom scope element that was mapped for your application, a custom security check defined in an adapter that is deployed to the same MobileFirst Server instance as your application, or a predefined MobileFirst security check.

For example, the following definition configures a mandatory application scope that contains the predefined application-authenticity security check (appAuthenticity) and a custom PincodeValidation scope element that was mapped for the application:

```
"mandatoryScope": "appAuthenticity PincodeValidation"
```

3. Deploy your copy of the application-descriptor JSON file to MobileFirst Server. See “Application configuration” on page 7-3.

To undo your configuration and eliminate the mandatory application scope, create a new copy of the application-descriptor file, and delete the `mandatoryScope` property definition or set the value to an empty string. Then redeploy the descriptor file to the server.

Results

After you successfully configure a mandatory application scope, you can see your defined mandatory application scope in the **Mandatory Application Scope** table on the application **Security** console page. In addition, you can see the mandatory-scope property definition in the application descriptor: in the console, go to the application **Configuration Files** tab. In the **Application-Descriptor JSON File** section you can see a copy of the application-descriptor JSON file. Search for the `mandatoryScope` property object in this file.

Mapping scope elements

Map custom scope elements to security checks to define application-specific security logic.

About this task

An OAuth scope is composed of zero or more scope elements, and each scope element is mapped to zero or more security checks (see OAuth scopes and security checks). You can define custom scope elements for your application, which map to any of the predefined or custom security checks that are available for the application.

The application scope mapping provides multiple advantages.

- Access the same resource from multiple applications, and customize the authorization logic of each application by using different maps for the same scope elements of the protecting resource scope.
- Reuse the same mandatory scope for multiple applications, and customize the authorization logic of each application by using different maps of the contained scope elements. See “Configuring a mandatory application scope” on page 7-276.
- Dynamically change the application's authorization logic by changing the scope-element maps. For example, you can define an empty scope element, and remap it to a new security check when the check becomes available.

Procedure

Map scope elements to security checks by using one of the following alternative methods:

- **Using IBM MobileFirst Platform Operations Console** (the console)

1. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Security** tab.
2. In the **Scope-Elements Mapping** section, select **Add to Scope**.
3. In the Add New Scope-Element Mapping dialog window, provide a name for the new element, select zero or more security checks to which to map the element, and then select **Add**. A scope-mapping table that reflects your configuration is displayed in the **Scope-Elements Mapping** section of the **Security** tab.
   Repeat this step as needed to map more scope elements.

You can delete or edit a defined scope element by selecting the relevant action icon for this element in the application's scope-mapping table.

- **Editing the application-descriptor file**
  1. Create a local copy of the application-descriptor JSON file. See "Application configuration" on page 7-3.
  2. Edit your local copy to define a `scopeElementMapping` object. In this object, define data pairs that are each composed of the name of your selected scope element, and a string of zero or more space-separated security checks to which the element maps. Replace *ScopeElement<n>* and *SecurityCheck<n>* with the names of the relevant scope element and security check:

```
"scopeElementMapping": {
    "ScopeElement1": "[SecurityCheck1 SecurityCheck2 ...]",
    ["ScopeElement2": "[SecurityCheck1 SecurityCheck2 ...]"
    ...]
}
```

     For example, the following code maps two scope elements:
     a. The `UserAuth` scope element is mapped to a custom UserAuthentication security check
     b. The `SSOUserValidation` scope element is mapped to the predefined LtpaBasedSSO security check, and to a custom CredentialsValidation security check.

```
"scopeElementMapping": {
    "UserAuth": "UserAuthentication",
    "SSOUserValidation": "LtpaBasedSSO CredentialsValidation"
}
```

  3. Deploy your copy of the application-descriptor JSON file to MobileFirst Server. See "Application configuration" on page 7-3.

  You can edit this definition at any time, as needed. To remove all scope-element mapping for your application, create a new copy of the application-descriptor file, delete the `scopeElementMapping` object, and redeploy the descriptor file to the server.

Results

After you successfully map one or more scope elements, you can see your defined scope elements in the **Scope-Elements Mapping** table on the application **Security** console page. In addition, you can see the scope-mapping property definition in the application descriptor: in the console, go to the application **Configuration Files** tab. In the **Application-Descriptor JSON File** section, you can see a copy of the application-descriptor JSON file. Search for the `scopeElementMapping` property definition in this file. This definition object contains one or more name/value data pairs of the following format:

```
"ScopeElement": "[SecurityCheck1 SecurityCheck2 ...]"
```

For example, the following code maps two scope elements:

1. The `UserAuth` scope element is mapped to a custom UserAuthentication security check
2. The `SSOUserValidation` scope element is mapped to the predefined LtpaBasedSSO security check, and to a custom CredentialsValidation security check.

```
"scopeElementMapping": {
    "UserAuth": "UserAuthentication",
    "SSOUserValidation": "LtpaBasedSSO CredentialsValidation"
}
```

Confidential clients

Learn how to allow confidential clients to connect to mobile services in a secure way. For example, you can grant a back-end service access to the Push service.

Overview

Confidential clients are clients that are capable of maintaining the confidentiality of their authentication credentials. You can use the MobileFirst authorization server to grant confidential clients access to protected resources, in accordance with the OAuth specification. This feature allows you to grant access to your resources to non-mobile clients, such as performance-testing applications. You begin by registering a confidential client with MobileFirst Server. As part of the registration, you provide the credentials of the confidential client, which consist of an ID and a secret. In addition, you set the client's allowed scope, which determines the scopes that can be granted to this client. When a registered confidential client requests an access token from the authorization server, the server authenticates the client by using the registered credentials, and verifies that the requested scope matches the client’s allowed scope.

Registering confidential clients

Register and manage confidential clients by using IBM MobileFirst Platform Operations Console (the console):

1. Select **Runtime Settings** in the console navigation sidebar, and then select the **Confidential Clients** tab.
2. Select **Create New** to register a new confidential client.
3. In the Create Confidential Client dialog window, provide the requested configuration parameters:
   - **`Display Name`** - an optional display name that is used to refer to the confidential client. The default display name is the value of the **`ID`** parameter.
   - **`ID`** - a unique identifier of the confidential client.
   - **`Secret`** - the secret that is used to authenticate the identity of the client.
   - **`Allowed Scope`** - the client's allowed scope. The scope is a space-separated list of scope elements. See OAuth scopes.
     An element of an allowed scope can also include the special asterisk wildcard character (*), which signifies any sequence of zero or more characters. For example, if the scope element is "send*", the confidential client can be granted access to scopes that contain any scope element that starts with "send", such as "sendMessage". The asterisk wildcard can be placed at any position within the scope element, and can also appear more than once.
     An allowed-scope parameter that consists of a single asterisk character (*) indicates that the confidential client can be granted a token for any scope.

4. Select **Save**. You can now see your new registered client in the confidential-clients table.
   You can delete clients or edit their registration information, at any time, by selecting the relevant action icon for the client entry in the table.

You might also see in the console's confidential-clients table the following preregistered MobileFirst confidential clients:

- "admin" and "push" - these clients are used for supporting push notifications from the administration service. These clients are automatically registered when the server starts with the push service enabled.

  Note: If you delete any of these clients, notifications from the administration service to the push service stop working until the server is restarted.
- "Test Client" - this client is preregistered with MobileFirst Development Server, and can be used for testing. The ID and secret of the Test Client confidential client are both "test", and the client has an unlimited allowed scope ("*").

Acquiring access tokens

To obtain an access token, the confidential client sends an access-token request with the `"client_credentials"` grant type, as described in the OAuth specification. The token request is an HTTP POST request that is sent to the URL of the token endpoint. The URL pattern for accessing the token endpoint is as follows (replace the <...> placeholders with your custom data):

```
http(s)://<server_ip>:<server_port>/<project_name>/api/az/v1/token
```

In the request, include the HTTP authorization header. The authorization server uses this header to authenticate the confidential client. Follow these steps to construct the authorization header:

1. Create a string that consists of the client ID and secret, separated by a colon (:). For example, for a `testClient` ID and a `testSecret` secret, use the string `testClient:testSecret`.
2. Encode the result string to Base64 format.
3. Add the string "Basic " before the encoded Base64 string.

For example, the authorization header for the client ID `testClient` and the client secret `testSecret` is formed in the following manner:

```
Authorization: Basic dGVzdENsaWVudDp0ZXN0U2VjcmV0
```

In addition to the authorization header, add the following two parameters to the request, using the `application/x-www-form-urlencoded` format:

- **`grant_type`** - this value must be set to `"client_credentials"`.
- **`scope`** - the requested scope, as a space-separated list of zero or more scope elements. See OAuth scopes.

When the authorization server receives the request, it authenticates the confidential client, and verifies that the requested scope is included in the client's allowed scope (as defined during registration). Each scope element in the requested scope must match one of the elements in the allowed scope (including wildcard expressions). If the request is accepted, the server issues an access token with the requested scope, and passes it to the client as a JSON object within the response. Following is an example of a response JSON object; (actual access tokens are longer than shown in the example):

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsImp3ay",
    "token_type": "Bearer",
    "expires_in": 3600,
    "scope": "sendMessage"
}
```

For more information about access tokens, see “Access tokens” on page 7-303. For information about the access-token response, see “Access-token response” on page 7-305.

Accessing protected resources

After the confidential client acquires an access token, it can use this token to access protected resources, such as adapter resources or MobileFirst Server endpoints. The client provides the access token by adding an HTTP authorization header to the HTTP request. The value of the header is constructed by inserting the string `"Bearer "` before the access token. Following is an example of a resource-request header:

```
Authorization: Bearer eyJhbGciOiJSUzI1NiIsImp3ay
```

Security checks

Learn how to create custom security checks, use the predefined MobileFirst security checks, and configure the behavior of your security checks at the adapter and application levels.

Security checks

Security checks constitute the basic server-side building block of the MobileFirst security framework. A security check is a server-side entity that implements a specific authorization logic. You protect a resource by assigning it a scope that maps to zero or more security checks. The security framework ensures that only a client that passes all of the security checks of the protecting scope is granted access to the resource. See “Overview of the MobileFirst security framework” on page 7-265. You can use security checks to authorize access both to resources that are hosted on MobileFirst Server and to resources on an external resource server. See “OAuth resource protection” on page 7-271.

A security check can be used to validate data from different sources, including
- Client data, such as login credentials (for example, user name and password, or a pin code), or application-authenticity data.
- Server-side state

Custom security checks are implemented and defined within MobileFirst adapters: the developer implements a security-check class in Java code, and configures it in the adapter descriptor. See “Security-checks implementation” on page 7-289.

The architecture of the security framework is modular and flexible. The implementation of the security check is not inherently dependent of any specific resource or application. You can reuse the same security check to protect different resources, and use different security-check combinations for various authorization flows. For enhanced flexibility, a security-check class exposes configuration properties that can be customized at the adapter level both in the security-check definition and during run time. You can also customize the configuration logic at the application level. See “Security-checks configuration” on page 7-297.

You can create custom security checks, and use any of the predefined MobileFirst security checks. See “Security-checks implementation” on page 7-289 and “Predefined MobileFirst security checks.”.

Predefined MobileFirst security checks

Learn about the predefined MobileFirst security checks.

Protect your resources by including any of the predefined security checks that are provided as part of IBM MobileFirst Platform Foundation in your custom OAuth scopes (see “OAuth resource protection” on page 7-271). You can also customize the configuration of the predefined security checks for a specific application (see “Configuring application security-check properties” on page 7-299). Proceed to the following topics for detailed information on each of the predefined security checks, and its configurable properties.

Application-authenticity security check:

Lean how to use the MobileFirst application-authenticity security check to validate the authenticity of your application and achieve enhanced resource protection.

Overview of MobileFirst application-authenticity validation

Use the predefined application-authenticity security check (appAuthenticity) to protect against unlawful attempts by fake or tampered applications to access your protected resources (APIs). When enabled, this check validates the authenticity of the application before providing it with any services. The application-authenticity security check is enabled by deploying to the server an application-authenticity file that you create with the MobileFirst application-authenticity tool: see “Enabling the application-authenticity security check.” The security framework uses this file to validate the authenticity of the application. By default, the security check is run during the application's runtime registration with MobileFirst Server, which occurs the first time an instance of the application attempts to connect to the server. However, as with any MobileFirst security check, you can also include this predefined check in custom security scopes: see OAuth scopes and security checks. For example, you can choose to add this check to the mandatory application scope: see “Configuring a mandatory application scope” on page 7-276.

The application-authenticity security check is supported for native iOS, native Android, native Windows 10 Universal Windows Platform and Windows 8 Universal, and cross-platform Cordova MobileFirst applications.

Proceed to the next topics to learn how to enable and configure the application-authenticity security check.

Enabling the application-authenticity security check:

Enable the predefined MobileFirst application-authenticity security check to protect against attempts by fake or tampered applications to access your resources (APIs).

About this task

You enable the application-authenticity security check by creating an application-authenticity file, and deploying the file to MobileFirst Server. You can select whether to separate the file creation and deployment steps, or consolidate them into one step:
- One-step authenticity-file generation and deployment with **mfpadm**

- Two-step authenticity-file generation and deployment

Procedure

-
- **One-step authenticity-file generation and deployment with `mfpadm`**

  Run the **`app version set authenticity-data`** command of the **`mfpadm`** command line program, or the **`<app-version> <set-authenticity-data>`** command through an **`mfpadm`** Ant task. Set the command's **`file`** argument or attribute to the location of your application binary file. This command will generate an application-authenticity file for your application, and store the file on the server.
- **Two-step authenticity-file generation and deployment**
  1. Get the MobileFirst application-authenticity Java tool, **`mfp-app-authenticity-tool.jar`**, by using either of the following alternative methods:
     - Download the tool from IBM MobileFirst Platform Operations Console (the console): from the console **Dashboard**, select **Download Center**, and then select the **Tools** tab. Under **Applicaiton-Authenticity Tool**, select **Download** and save the file to your preferred location.
     - Copy the tool from the <*product_install_dir*>`/MobileFirstServer/external-server-libraries/` directory (where <*product_install_dir*> is the directory in which you installed IBM MobileFirst Platform Foundation).
  2. Generate a unique application-authenticity file: from the command line, run the application-authenticity tool with one of the following command variations:
     -
       ```
       java -jar <path to mfp-app-authenticity-tool.jar>
           <app_binary> [<authenticity_file>]
       ```
     - `java -jar <path to mfp-app-authenticity-tool.jar>`

     When no parameters are provided, the application-authenticity tool runs in an interactive mode. You are then prompted to enter the path to your application binary file (*app_binary*), and optionally also the path to your target application-authenticity file (*authenticity_file*).

     `mfp-app-authenticity-tool` parameters

     app_binary

     Mandatory path to your application binary file.
     - For Android, refer to your `.apk` application file. This file must be signed. For more information about signing Android applications, see the Android documentation: Signing Your Applications.
       Note that the Google Play multiple APK feature cannot be used together with the MobileFirst application-authenticity validation. For information about multiple APK, see the Android documentation: Multiple APK Support.
     - For iOS, refer to your `.ipa` application 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.
       Note that bitcode-enabled applications cannot be used together with the MobileFirst application-authenticity validation. See "Working with bitcode in iOS apps" on page 7-48.
     - For Windows 10 Universal Windows Platform (UWP) and Windows 8.1 Universal, refer to your `.appx` application file, or to a `.appx` file from a bundle.

     authenticity_file

     Optional path to the generated application-authenticity file. By default, the tool generates an *<application-binary base file name>*.authenticity_data file in the same directory as the provided application binary file (*app_binary*).

Example
The following command is run from the directory that contains the application-authenticity tool, and does not set the optional *authenticity_file* parameter. The command generates a my_ios_app.authenticity_data application-authenticity file in the same directory as the input my_ios_app.ipa application binary: /Users/myname/.

```
java -jar mfp-app-authenticity-tool.jar /Users/myname/my_ios_app/my_ios_app.ipa
```

3. Deploy your generated application-authenticity file to MobileFirst Server, by using either MobileFirst Operations Console or **mfpadm**:
   - In the console,
     a. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Authenticity** tab.
     b. Select **Upload Authenticity File**, browse to your generated application-authenticity file, and upload the file.
   - Run the **app version set authenticity-data** command of the **mfpadm** command line program, or run the **<app-version> <set-authenticity-data>** command through an **mfpadm** Ant task. Set the command's **file** argument or attribute to the location of your application-authenticity data file.

   When your application-authenticity file is successfully deployed to the server, a relevant message is displayed in the console.

Results

When your application-authenticity file is deployed to the server, the **Status** value in the application **Authenticity** console tab is set to "Enabled", indicating that the security check is enabled for your application.

You can retrieve a copy of the application-authenticity file that is deployed for your application on the server, by running the **app version get authenticity-data** command of the **mfpadm** command line program, or the **<app-version> <get-authenticity-data>** command through an **mfpadm** Ant task.

You can disable the application-authenticity security check at any time, by using one of the following methods:
- In the application **Authenticity** console tab, select **Delete Authenticity File**.
- Run the **app version delete authenticity-data** command of the **mfpadm** command line program, or the **<app-version> <delete-authenticity-data>** command through an **mfpadm** Ant task.

Configuring the application-authenticity security check:

Configure the predefined MobileFirst application-authenticity security check to match your specific requirements.

About this task

The predefined application-authenticity security check (appAuthenticity) has a single configurable property: **expirationSec**. This property sets the expiration period for a successful security-check state. The expiration period determines the minimal interval for invoking the check again after a successful execution.

You can configure the value of the expirationSec application-authenticity security-check property, as outlined in the following procedure.

Note: The procedure explains how to use IBM MobileFirst Platform Operations Console to configure the property value. Alternatively, you can also set the property value directly in the application-descriptor file. For detailed information, see "Configuring application security-check properties" on page 7-299.

Procedure

1. In MobileFirst Operations Console, select your application version from the **Applications** section of the navigation sidebar, and then select the application **Security** tab.
2. In the **Security-Check Configurations** section, select **Create New**.
3. In the Configure Security-Check Properties window, configure the application-authenticity security check:
   a. In the **Security Check** field, select appAuthenticity from the list.
   b. In the **Expiration Period, Successful State (seconds)** field, set your preferred expiration period for a successful state of the security check, in seconds.

   When the configuration is done, you can see and edit your appAuthenticity security-check configuration in the **Security-Check Configurations** table of the application **Security** tab. See "Configuring application security-check properties" on page 7-299.

LTPA-based single sign-on (SSO) security check:

Learn how to use the MobileFirst LTPA-based SSO security check to use a back-end service to authenticate users by using SSO LTPA tokens.

See "The MobileFirst LTPA-based SSO security check" on page 7-288 and "Configuring the LTPA-based SSO security check" on page 7-289.

LTPA Overview

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.

Figure 7-32 on page 7-286 shows a simple client <-> server flow with LTPA.

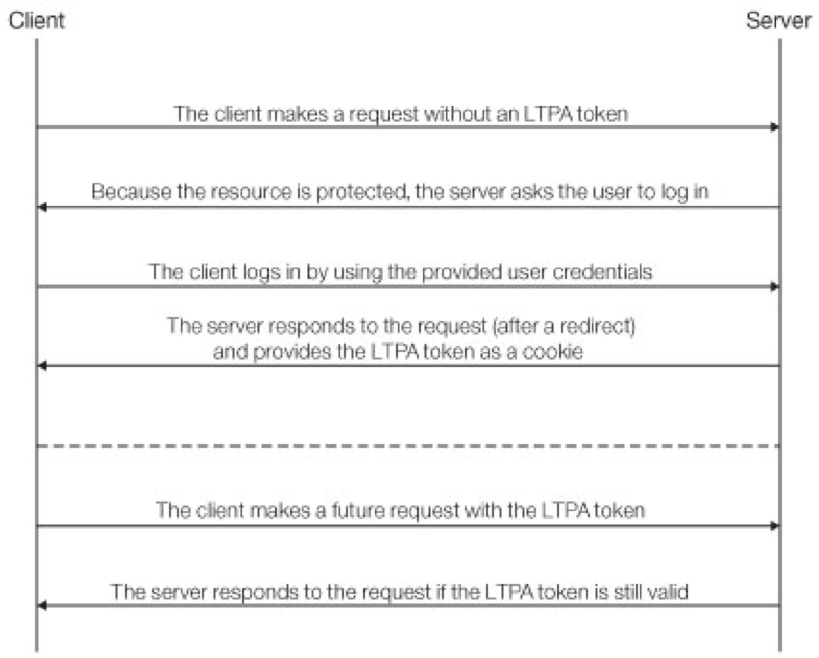

Client
Server
The client makes a request without an LTPA token
Because the resource is protected, the server asks the user to log in
The client logs in by using the provided user credentials
The server responds to the request (after a redirect) and provides the LTPA token as a cookie
The client makes a future request with the LTPA token
The server responds to the request if the LTPA token is still valid

Figure 7-32. Simple LTPA-based client <-> server flow

After a user logs in to the server, 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, the need for subsequent user interaction is avoided.

LTPA tokens have a configurable expiration time to reduce the possibility of session hijacking.

Reverse proxy with LTPA

Your infrastructure can also use the LTPA token to communicate with a back-end server that acts on behalf of the user. In a reverse-proxy topology, the user cannot directly access the back-end server. The reverse proxy can be used to authenticate a user's identity, and then send the LTPA token of the authenticated user to back-end servers. This configuration ensures that access to MobileFirst Server cannot be obtained until a user is authenticated. This is useful, for example, when you do not want to use IBM MobileFirst Platform Foundation to handle vital user credentials, or when you want to use an existing authentication setup. Enterprise environments should use a reverse proxy, such as IBM WebSphere DataPower or IBM Security Access Manager, in the DMZ, and place the MobileFirst Server in the intranet.

In a reverse-proxy implementation, MobileFirst Server must be configured for LTPA authentication to get the user identity.

Figure 7-33 on page 7-287 shows an LTPA flow between a client and a back-end server using a reverse proxy.

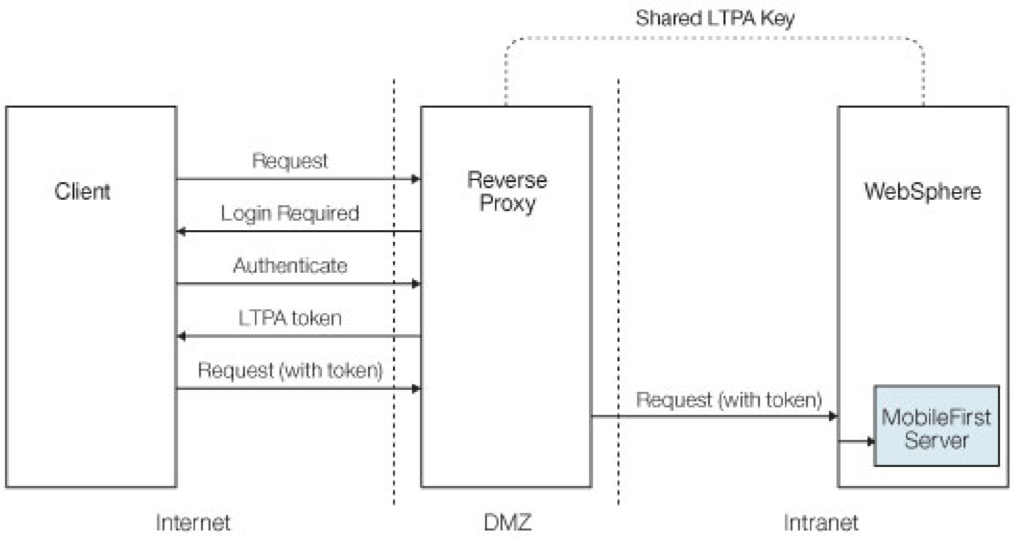


Figure 7-33. Reverse-proxy LTPA flow

MobileFirst integration with a reverse proxy

You can use a reverse proxy to enable enterprise connectivity within a MobileFirst environment, and to provide authentication services to IBM MobileFirst Platform Foundation.

General architecture

Reverse proxies typically front MobileFirst Server instances as part of the deployment, as shown in Figure 7-34, and follow the gateway pattern.

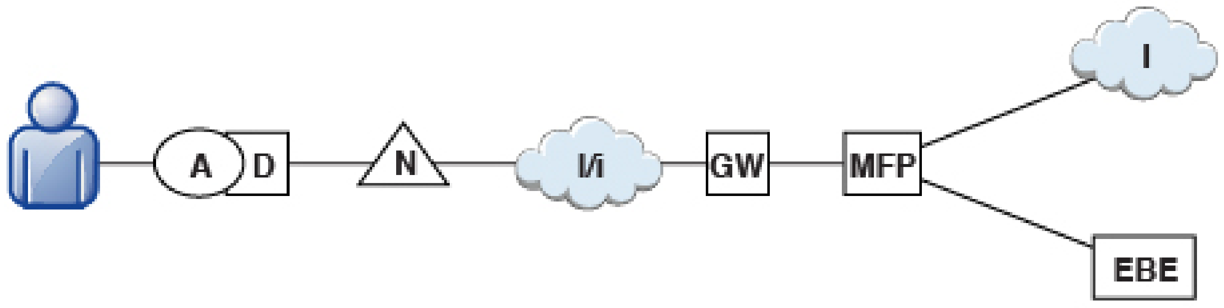


Figure 7-34. Integration with reverse proxy

The MFP icon represents an instance of MobileFirst Server. The GW icon represents a reverse-proxy gateway, such as WebSphere DataPower. 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 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 to use the user identity from one of these mechanisms, and establish a successful log in. Figure 7-35 shows a typical authentication flow for this gateway topology.

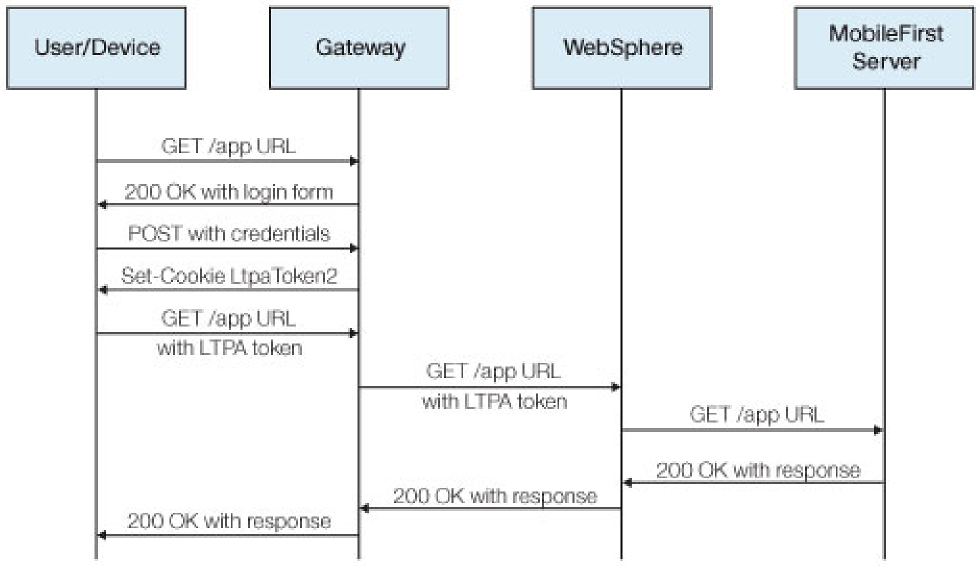


Figure 7-35. Authentication flow

This configuration was successfully tested with WebSphere DataPower for LTPA-based authentication. On successful authentication, the gateway forwards an LTPA token (in the form of an HTTP cookie) to WebSphere Application Server, which validates the LTPA token and creates a caller principal. IBM MobileFirst Platform Foundation can use this caller principal, as needed.

The MobileFirst LTPA-based SSO security check

The predefined MobileFirst LTPA-based single-sign on (SSO) security check (LtpaBasedSSO) enables integration of IBM MobileFirst Platform Foundation with the WebSphere Application Server LTPA protocol. This security check allows you to integrate instances of MobileFirst Server within an LTPA-based gateway topology, as described in the previous sections, and use a back-end service to authenticate users by using an SSO LTPA token.

This predefined security check can be used as any other security check in the MobileFirst security framework (see “Security checks” on page 7-281): you can map a custom scope element to this check, and use the check (or a scope element that contains it) in a protecting resource scope or in a mandatory application scope. See “OAuth resource protection” on page 7-271.

You can also configure the behavior of this security check for your application, as outlined in the next topic.

Configuring the LTPA-based SSO security check:

Learn how to configure the predefined MobileFirst LTPA-based single sign-on (SSO) security check.

About this task

The predefined LTPA-based single sign-on (SSO) security check (LtpaBasedSSO) has a single configurable property: `expirationSec`. This property sets the expiration period for a successful security-check state. The expiration period determines the minimal interval for invoking the check again after a successful execution.

You can configure the value of this expirationSec property, as outlined in the following procedure.

Note: The procedure explains how to use the IBM MobileFirst Platform Operations Console to configure the property value. Alternatively, you can also set the property value directly in the application-descriptor file. For detailed information, see “Configuring application security-check properties” on page 7-299.

Procedure

1. Open a MobileFirst Operations Console window. Select your application version from the **Applications** section of the navigation sidebar, and then select the application **Security** tab.
2. In the **Security-Check Configurations** section, select **Create New**.
3. In the Configure Security-Check Properties window, configure the LTPA-based SSO security check:
   a. In the **Security Check** field, select LtpaBasedSSO from the list.
   b. In the **Expiration Period, Successful State (seconds)** field, set your preferred expiration period for a successful state of the security check, in seconds.

   When the configuration is done, you can see and edit your LtpaBasedSSO security-check configuration in the **Security-Check Configurations** table of the application **Security** tab. See “Configuring application security-check properties” on page 7-299.

Security-checks implementation

Learn how to implement security checks that provide custom authorization logic.

Overview

The development of a security check involves the following server-side steps:

1. Create a security-check class that implements the security-check interface (SecurityCheck). For more information about the requirements of this class, see “The security-check contract” on page 7-291. You can start your development by extending one of the provided security-check base classes. See “The security-check base and sample classes” on page 7-290.
2. Optionally create a security-check configuration class that implements the security-check configuration interface (SecurityCheckConfiguration). You can start with the abstract implementation of this interface, the SecurityCheckConfigurationBase class, or with one of the provided sample

implementations that extend this class. For more information, see "The security-check contract" on page 7-291 and "The security-check base and sample classes."

3. Define one or more security checks of a custom security-check class type. See "Defining security checks" on page 7-294.

Note:

- The MobileFirst security framework requires that you implement a custom security check as part of an adapter that is deployed to MobileFirst Server. You implement the security-check class by using the MobileFirst security server-side Java API, and you define an instance of this class in the adapter-descriptor file (`adapter.xml`). You can implement and define security checks either in the same adapter that defines your resources, or in a separate dedicated adapter, as you prefer.
- The outlined steps do not need to be executed in the specified order, and they can be done in stages. For example, you can define an empty security-check definition, and add configuration properties when the related security-check configuration is ready. But be aware of the following considerations:
  - To deploy an adapter that defines a security check, the security check's class must be available in the same adapter, either as part of the adapter source code or as via an external library.
  - To correctly define the configuration properties in the security-check definition, you need to know which properties are supported for the referenced class and what are their value restrictions.

After you define a security-check class and deploy it to MobileFirst Server, you can customize the value of its properties both for the specific server instance and for a specific application version. See "Configuring runtime adapter security-check properties" on page 7-298 and "Configuring application security-check properties" on page 7-299. The administrator can edit these configurations before going to production, and after the application is already in production.

The security-check base and sample classes

To facilitate and accelerate your development process, IBM MobileFirst Platform Foundation provides base abstract implementations of the SecurityCheck interface. In addition, a base abstract implementation of the SecurityCheckConfiguration interface is provided (SecurityCheckConfigurationBase), as well as complementary sample security-check configuration classes for each of the provided base security-check classes. Start out with the base security-check implementation (and related sample configuration) that best fits your development needs, and extend and modify the implementation as needed.

ExternalizableSecurityCheck

This class implements the required externalization of the security check as a JSON object, and also implements a security-check state mechanism.

ExternalizableSecurityCheck creates a security-check configuration of the sample ExternalizableSecurityCheckConfig class.

CredentialsValidationSecurityCheck

This class extends the ExternalizableSecurityCheck class and adds an implementation that validates user credentials as a condition for accessing a protected resource. The implementation allows a limited number of login attempts during a certain interval, after which the security check is blocked for a configured period. In the case of a successful login, the state of the security check remains successful for a configured period, during which the user can access the requested resource.

CredentialsValidationSecurityCheck creates a security-check configuration of the sample CredentialsValidationSecurityCheckConfig class, which extends ExternalizableSecurityCheckConfig and defines the configurable properties of the security check and their default values.

For guidelines on how to implement and configure the CredentialsValidationSecurityCheck security check, and how to implement complementary client-side challenge handlers, see the CredentialsValidationSecurityCheck tutorials.

UserAuthenticationSecurityCheck
This class extends the CredentialsValidationSecurityCheck class and adds to it an implementation that creates a user identity that can be used to identify the current user. The class also implements a sample "remember me" function, which uses a user identify that is stored in the registration service as the active user.

UserAuthenticationSecurityCheck creates a security-check configuration of the sample UserAuthenticationSecurityCheckConfig class, which extends CredentialsValidationSecurityCheckConfig.

For guidelines on how to implement and configure the UserAuthenticationSecurityCheck security check, and how to implement complementary client-side challenge handlers, see the UserAuthenticationSecurityCheck tutorials.

The ExternalizableSecurityCheck and ExternalizableSecurityCheckConfig classes are included in the com.ibm.mfp.server.security.external.checks.impl package of the core MobileFirst server-side Java API.
The CredentialsValidationSecurityCheck, CredentialsValidationSecurityCheckConfig, UserAuthenticationSecurityCheck, and UserAuthenticationSecurityCheckConfig classes are available as part of the MobileFirst com.ibm.mfp.security.checks.base Java Maven library, which you can download from the Maven repository or from the IBM MobileFirst Platform Operations Console: from the console **Dashboard**, select **Download Center**, select the **Tools** tab, and choose the **Download** option in the **Security Checks** section.

The security-check contract:

Learn about the MobileFirst security-check contract, which is defined by the SecurityCheck and SecurityCheckConfiguration interfaces.

Overview

Every security check must implement the com.ibm.mfp.server.security.external.SecurityCheck interface (*the security-check interface*). This interface constitutes the basic contract between the security check and the MobileFirst security framework. Custom security checks are implemented as a Java security-check class within a MobileFirst adapter (see “Security-checks implementation” on page 7-289). The security-check implementation must fulfill the following requirements:

- Functions - the security check must provide the client-authorization and introspection functions. See “Security-check functions” on page 7-292.

- State management - the security check must manage its state, including creation, disposal, and current-state management. See "Security-check state management" on page 7-293.
- Configuration - the security check must create a security-check configuration object, which defines the supported security-check configuration properties, and validates the types and values of customizations of the basic configuration. See "Security-check configuration" on page 7-293.

For a complete reference of the SecurityCheck interface, see the code documentation of this interface. In addition, review the implementation and documentation of the provided abstract security-check base classes. These classes implement some of the requirements of the security-check contract, such as state management, and demonstrates how to implement other custom functions. See "The security-check base and sample classes" on page 7-290.

Security-check functions

A security check provides two main functions to the security framework:

Authorization

The framework uses the SecurityCheck.authorize method to authorize client requests. When the client requests access to a specific OAuth scope, the framework maps the scope elements into security checks (see OAuth scopes and security checks). For each security check in the scope, the framework calls the authorize method to request authorization for a scope that contains the scope elements that mapped to this security check. This scope is provided in the method's `scope` parameter. The security check adds its response to the AuthorizationResponse object that is passed to it within the `response` parameter. The response contains the name of the security check and the response type, which can be success, failure, or a challenge (see AuthorizationResponse.ResponseType). When the response contains a challenge object or custom success or failure data, the framework passes the data to the client's security-check challenge handler within a JSON object. For success, the response also contains the scope for which the authorization was requested (as set in the `scope` parameter), and the expiration time for the granted authorization. To grant the client access to the requested scope, the authorize method of each of the scope's security checks must return success, and all expiration times must be later than the current time.

Introspection

The framework uses the SecurityCheck.introspect method to retrieve introspection data for a resource server. This method is called for each security check that is contained in the scope for which introspection was requested. As with the authorize method, the introspect method receives a `scope` parameter that contains the scope elements that mapped to this security check. Before returning the introspection data, the method verifies that the current state of the security check still supports the authorization that was previously granted for this scope. If the authorization is still valid, the introspect method adds its response to the IntrospectionResponse object that is passed to it within the `response` parameter. The response contains the name of the security check, the scope for which the authorization was requested (as set in the `scope` parameter), the expiration time for the granted authorization, and the requested custom introspection data. If authorization can no longer be granted (for example, if the expiration time for a previous successful state elapses), the method returns without adding a response.

Note:

- The security framework collects the processing results from the security checks, and passes relevant data to the client. The framework processing is entirely ignorant of the states of the security checks.
- Calls to the authorize or introspect methods can result in a change in the current state of the security check, even if the expiration time of the current state did not elapse.

Security-check state management

Security checks are stateful, meaning that the security check is responsible for tracking and retaining its interaction state. On each authorization or introspection request, the security framework retrieves the states of relevant security checks from external storage (usually, distributed cache). At the end of request processing, the framework stores the security-check states back in external storage.

The security check contract requires that a security check

- Implement the java.io.Externalizable interface. The security check uses this interface to manage the serialization and deserialization of its state.
- Define an expiration time and an inactivity timeout for its current state. The state of the security check represents a stage in the authorization process, and cannot be indefinite. The specific periods for the state's validity and maximum inactivity time are set in the security-check implementation, according to the implemented logic. The security check informs the framework of its selected expiration time and inactivity timeout via the implementation of the getExpiresAt and getInactivityTimeoutSec methods of the SecurityCheck interface.

Security-check configuration

A security check can expose configuration properties, whose values can be customized both at the adapter and at the application level. The security-check definition of a specific class determines which of the supported configuration properties of this class to expose, and can customize the default values set in the class definition. See “Defining security checks” on page 7-294. The property values can be further customized, dynamically, both for the adapter that defines the security checks, and for each application that uses the check. See “Security-checks configuration” on page 7-297. A security-check class exposes its supported properties by implementing a createConfiguration method, which creates an instance of a security-check configuration class that implements the com.ibm.mfp.server.security.external.SecurityCheckConfiguration interface (*the security-check configuration interface*). This interface complements the SecurityCheck interface, and is also part of the security-check contract. The security check can create a configuration object that does not expose any properties, but the createConfiguration method must return a valid configuration object and cannot return `null`. For a complete reference of the SecurityCheckConfiguration interface, see the code documentation of this interface. In addition, review the implementation and documentation of the provided abstract security-check base class and the sample configuration-class implementations. See “The security-check base and sample classes” on page 7-290.

The security framework calls the security-check's createConfiguration method during deployment, which occurs for any adapter or application configuration change. The method's **`properties`** parameter contains the properties that are defined in the adapter's security-check definition, and their current customized values (or the default value if there was no customization). The implementation of the security-check configuration should validate the values of the received properties, and provide methods for returning the validation results. The security-check configuration must implement getErrors, getWarnings, and getInfo methods. The abstract security-check configuration base class, SecurityCheckConfigurationBase also defines and implements custom getStringProperty, getIntProperty, and addMessage methods. See the code documentation of this class for details.

Note: The names and values of the configuration properties in the security-check definition and in any adapter or application customization, must match the supported properties and allowed values, as defined in the configuration class.

Defining security checks:

Learn how to define custom security checks.

Before you begin

Ensure that the security-check class that you want to use in your definition is available in your adapter project, either as part of the source code or via an external library. See “Security-checks implementation” on page 7-289.

About this task

A security check is an instance of a security-check class, which is defined in the adapter descriptor. The defined security check can be used within a security scope to apply a specific resource-protection logic. Follow the outlined procedure to define a custom security check:

Procedure

1. Add a security-check definition: in the adapter-descriptor file (`adapter.xml`), add a <securityCheckDefinition> element of a security-check class that is available in your adapter project. For a detailed reference of the security-check definition element and usage guidelines, see “The <securityCheckDefinition> element” on page 7-295.
2. To apply your changes and make your security check available for inclusion in security scopes, build your adapter and deploy it to an instance of MobileFirst Server (the server). See “Working with Java adapters” on page 7-196 and “Working with JavaScript adapters” on page 7-214.

Results

After you successfully deploy an adapter with a security-check definition to the server, this security check can be used within security scopes and scope elements of any adapter or application that are deployed or registered to the same server instance. See “OAuth resource protection” on page 7-271.

You can also see your security check and its configuration information, and make runtime configuration changes, from IBM MobileFirst Platform Operations Console (the console):

Note: When deploying an adapter during an active console session, you need to refresh the console page to reflect your changes.

- Select your adapter from the **Adapters** section of the console's navigation sidebar, and then select the adapter **Configuration Files** tab. In the **Adapter-Descriptor XML File** section, you can see the server copy of your adapter descriptor, including the <securityCheckDefinition> element that defines your custom security check and its configurable properties.
- Select the **Security Checks** tab for your adapter. Search for the name of your security check, as set in the **`name`** attribute of your security-check definition element (<securityCheckDefinition>). You can see a list of all the configuration properties that you exposed in the security-check definition. The properties are referenced by the value of their configured **`displayName`** attribute, or by the value of the **`name`** attribute when no display name is configured. If you set the property's **`description`** attribute in the definition, this description is also displayed.
  For each property, the value that is configured in the **`defaultValue`** attribute is shown as the current value. You can change the value to override the default value from your security-check definition. You can also restore, at any time, the original default values from your security-check definition.
  You can modify the property values on this page to customize the security-check configuration for this specific MobileFirst Server instance. See "Configuring runtime adapter security-check properties" on page 7-298.
- Select an application version from the **Applications** section of the console's navigation sidebar (provided at least one application is registered with this instance of the server). Then select the application **Security** tab. If you choose to map scope elements or define a mandatory application scope, you can select your security check from among the custom security checks: see "Mapping scope elements" on page 7-277 and "Configuring a mandatory application scope" on page 7-276. If you choose to configure security-check properties, you can see your defined properties and their descriptions (if provided). You can also see the default property values, as set it the security-check definition or overwritten in the adapter runtime configuration. See "Configuring application security-check properties" on page 7-299.

The <securityCheckDefinition> element:

Learn how to use the <securityCheckDefinition> adapter-descriptor element to define a custom security check.

Overview

You define a security check by adding a <securityCheckDefinition> XML element within the <mfp:adapter> element of your adapter-descriptor file (`adapter.xml`). Each security check is an instance of a security-check class. See "Defining security checks" on page 7-294. The security-check definition can contain zero or more <property> subelements that represent configurable security-check properties. Following is a reference of the <securityCheckDefinition> element, its attributes, and subelements.

Syntax

```
<securityCheckDefinition name="securityCheckName"
                         class="securityCheckClass">
   <property name="propertyName" displayName="propertyDisplayName"
             defaultValue="defaultPropertyValue"
             description="propertyDescription>"/>
</securityCheckDefinition>
```

Attributes

The <securityCheckDefinition> element accepts the following mandatory attributes:

`name` The name of the defined security check.

`class` The type of the security check, as a full path to a security-check class implementation that is available in the same adapter as the definition (either as source code or via an external library). For example, `com.my_company.package.SampleSecurityCheck` for a SampleSecurityCheck class that is implemented in a com.my_company.package package.

The <property> subelement

The security-check definition can contain <property> elements for any configuration property that is supported by the security-check's class. The supported configurations are defined in the security-check configuration class (of the SecurityCheckConfiguration interface type) that is created by the security-check class. When defining your security check, you can decide which of the supported properties to expose, and override the configuration class's default property values. For supported properties that are not referenced in the security-check definition, the security check relies on the default configuration-class values.
The properties that you expose in the security-check definition can be further customized, at run time, both at the adapter level and at the application level.

<property> attributes
The <property> subelement of the <securityCheckDefinition> element accepts the following attributes:

Mandatory attributes

`name` The name of a security-check configuration property that is supported by the class of the security-check definition (via its security-check configuration class).

`defaultValue`
The default value to use for this property. This value overrides the default value set in the related security-check configuration class.

Optional attributes

`displayName`
The display name to use for this property. MobileFirst Operations Console uses the display name when referencing the property. The default display name is the value of the **`name`** attribute.

`description`
A textual description of the property and its purpose. When provided, MobileFirst Operations Console displays the description as a hint for fields that contain the property's value.

Note: For the purposes of using the console, you do not need to include the default property value in the description string. The console displays the default-value information based on the value of the **`defaultValue`** attribute.

Example

The following example defines a UserAuthenticationSC security check of a custom MyUserAuthenticationSecurityCheck security-check class, which is implemented in

the com.my_company.package Java package.
The custom security-check class extends the sample MobileFirst abstract UserAuthenticationSecurityCheck base class, and creates a custom MyUserAuthenticationSecurityCheckConfiguration class that extends the sample UserAuthenticationSecurityCheckConfig class. The custom security check inherits all the configuration properties of the extended sample class and its ancestor classes (CredentialsValidationSecurityCheckConfig and ExternalizableSecurityCheckConfig): `inactivityTimeoutSec` (default value = `0`), `maxAttempts` (default value = `1`), `attemptingStateExpirationSec` (default value = `120`), `successStateExpirationSec` (default value = `3,600`), `failureStateExpirationSec` (default value = `0`), `rememberMeDurationSec` (default value = `0`).In addition, the custom configuration class defines a `pinCode` property (default value = `1234`).
The custom security-check definition exposes only the `pinCode`, `maxAttempts`, `attemptingStateExpirationSec`, and `failureStateExpirationSec` properties. Of these properties, it customizes the default values of the `pinCode`, `maxAttempts`, and `failureStateExpirationSec` properties, changing them to `9876`, `3`, and `180`.

```
<securityCheckDefinition name="UserAuthenticationSC" class="com.my_company.package.MyUserAuthenticationSecurityCheck">
    <property name="pinCode" displayName="Pin Code"
              defaultValue="9876"
              description="A four-digit pin code"/>
    <property name="maxAttempts" displayName="Maximum attempts"
              defaultValue="3"
              description="Maximum allowed user-authentication attempts"/>
    <property name="attemptingStateExpirationSec" displayName="Expiration Period, Attempting State (seconds)"
              defaultValue="120"
              description="Expiration period for an attepmpting security-check state, in seconds"/>
    <property name="failureStateExpirationSec" displayName="Expiration Period, Failed State (seconds)"
              defaultValue="180"
              description="Expiration period for a failed security-check state, in seconds"/>
</securityCheckDefinition>
```

Security-checks configuration

Learn about the security-check configuration hierarchy, and how to configure security-check properties at the adapter and at the application levels.

The definition of a predefined or custom security check exposes zero or more configuration properties. The documentation of the predefined security checks lists the properties that are supported for each check, and their default values: see "Predefined MobileFirst security checks" on page 7-282. For custom security checks, the configuration properties are exposed in the definition of the security check in the adapter descriptor, which also sets the default property values: see "Defining security checks" on page 7-294. The default values of custom security-check configuration properties can be customized, at the adapter level, for a specific MobileFirst Server instance. The values of both custom and predefined security-check configuration properties can be further customized for a specific application version. See the detailed customization instructions in the following topics:

- "Configuring runtime adapter security-check properties" on page 7-298
- "Configuring application security-check properties" on page 7-299

Note:
- The security-check definition, which contains the basic configuration, is defined in the XML descriptor file of the adapter that defines the security check.
- Runtime adapter customizations of the defined configuration properties for a specific server instance are defined in the adapter runtime-configuration JSON file.

- Application customizations of the security-check configuration properties are defined in the application-descriptor JSON file.

Configuring runtime adapter security-check properties:

Learn how to configure adapter security-check properties for a specific MobileFirst Server instance.

About this task

The definition of a custom security check exposes zero or more configuration properties, and defines their default values. You can see the security-check definitions of an adapter that is deployed to MobileFirst Server in the server's copy of the adapter-descriptor XML file. See "Defining security checks" on page 7-294. The security-check configuration that is set in the definition applies to all instances of MobileFirst Server to which you deploy the adapter that defines the security check. Follow the outlined procedure to dynamically customize the security-check configuration for a specific instance of MobileFirst Server, without changing the original security-check definition, or having to redeploy the adapter.

Note: Runtime adapter customizations of the security-check configuration properties are defined in the adapter runtime-configuration JSON file.

Procedure

Customize the adapter configuration of your selected security check for a specific instance of MobileFirst Server by using one of the following methods:

- **Using IBM MobileFirst Platform Operations Console** (the console)
  1. In the **Adapters** section of the console's navigation sidebar, select the adapter that defines the security checks that you want to configure, and then select the **Security Checks** tab. You can see a list of all the security checks that are defined in the selected adapter. For each security check, you can select **View** to see a list of the security check's properties and their current values, the default values from the security-check definition (when the default differs from the current value), and the property description (if provided in the definition).
  2. Change the values of the properties that you want to customize for this MobileFirst Server instance. Then select **Save** at the end of the security-check's properties list.
     You can always restore the original property configuration values of the security-check definition by selecting **Restore Default Values**.
- **Editing the adapter runtime-configuration file**
  1. Create a local copy of the adapter runtime-configuration JSON file. You can use either of the following methods to copy the content of the file, and then paste it into a local file:
     - In the **Adapters** section of the MobileFirst Operations Console navigation sidebar, select the adapter that defines the security checks that you want to configure, and then select the **Configuration Files** tab. The content of the runtime-configuration file is displayed in the **Adapter Runtime-Configuration JSON File** section. You can use the file-copy icon next to the displayed file to copy the content.
     - Run the `show user-config` command of the `mfpadm` command-line program or Ant task.
  2. In your local copy of the configuration file, look for a `securityCheckDefinitions` object within the adapter object. If the object does

not exist, create it. In this object, find or create an object that is named as your selected security check (*SecurityCheckName* in the following template):

```
"securityCheckDefinitions": {
    "SecurityCheckName": {
        }
    }
}
```

3. In your security-check object (*SecurityCheckName*), find or add a `properties` object. For each available configuration property that you want to configure, add within the `properties` object a pair of configuration-property name and value:

```
"securityCheckDefinitions": {
    "SecurityCheckName": {
        "properties": {
            "property1Name": "property1Value",
            ["property2Name": "property2Value",
             ...]
        }
    }
}
```

Example
The following example sets the values of the **`maxAttempts`** and **`failureExpirationSec`** properties of a custom UserAuthenticationSC security check to `4` and `90`:

```
"securityCheckDefinitions": {
    "UserAuthenticationSC": {
        "properties": {
            "maxAttempts": "4",
            "failureExpirationSec: "90"
        }
    }
}
```

4. Deploy your copy of the adapter runtime-configuration JSON file to MobileFirst Server. You can do this by running the **`set user-config`** command of the **`mfpadm`** command-line program or Ant task.

You can repeat this procedure, at any time, to customize the security-check configuration. You can also deploy the same configuration file to other instances of MobileFirst Server on which the same adapter is deployed, or reuse relevant portions of the configuration in other adapter configuration files.

Results

After completing the configuration changes, you can see your defined property values in the console, both in the adapter **Security Checks** page and in the **Adapter Runtime-Configuration JSON File** section on the adapter **Configuration Files** page.
If you select to customize the configuration of the same security check for a specific application, the console displays your customized property values as the default values for the application configuration. See “Configuring application security-check properties.”

Configuring application security-check properties:

Learn how to customize the security-check configurations for a specific application version.

About this task

You can make application-specific changes to the default values of any predefined or custom security-check property that is exposed on the same MobileFirst Server instance as your application. The documentation of the predefined security checks lists the properties that are supported for each check, and their default values. See "Predefined MobileFirst security checks" on page 7-282. For custom security checks, the basic configuration is defined in the adapter descriptor file, and can be overridden for a specific server instance in the adapter runtime-configuration file. See "Security-checks configuration" on page 7-297. In addition, for the custom security checks the MobileFirst security framework provides an application-specific property for enabling device SSO. See "Configuring device single sign-on (SSO)" on page 7-301. The IBM MobileFirst Platform Operations Console for your MobileFirst Server instance displays the available security checks and their properties, including the property values, default values, and descriptions (if provided in the definition). Follow the outlined procedure to customize the property values for your application.

Note: Application customizations of the security-check configuration properties are defined in the application-descriptor JSON file. See "Application configuration" on page 7-3.

Procedure

Configure the security checks that are used by your application by using one of the following alternative methods:

- **Using IBM MobileFirst Platform Operations Console** (the console)
  1. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Security** tab.
  2. In the **Security-Check Configurations** section, select **Create New**, or select the edit icon for an existing security-check configuration (if exists).
  3. In the Configure Security-Check Properties dialog window, select the security check that you want to configure from among the displayed list of available predefined and custom security checks. The dialog window displays a list of the supported properties of your selected security check, their current values, the default values (if they differ from the current values), and their descriptions (if provided). Edit the values that you want to change, and select **OK** to submit your changes.

     Note: In some cases, the dialog window spawns multiple pages. Use the arrow keys to change pages and see all the supported properties.

  You can delete or edit your security-check configuration, at any time, by selecting the relevant action icon for your security check in the security-check configurations table.
- **Editing the application-descriptor file**
  1. Create a local copy of the application-descriptor JSON file. See "Application configuration" on page 7-3.
  2. In your local copy of the descriptor file, look for a `securityCheckConfigurations` object. If the object does not exist, create it. In this object, find or create an object that is named as your selected security check (*SecurityCheckName* in the following template). Within the security-checks object, add a pair of configuration-property name and value for each available configuration property that you want to configure:

```
"SecurityCheckConfigurations": {
    "SecurityCheckName": {
        "property1Name": "property1Value",
        ["property2Name": "property2Value",
         ...]
      }
    }
}
```

Example

The following example sets the values of the **maxAttempts** and **failureExpirationSec** properties of a custom UserAuthenticationSC security check to 2 and 60:

```
"SecurityCheckConfigurations": {
    "UserAuthenticationSC": {
        "properties": {
            "maxAttempts": "2",
            "failureExpirationSec: "60"
        }
    }
}
```

3. Deploy your copy of the application-descriptor JSON file to MobileFirst Server. See “Application configuration” on page 7-3.

You can repeat this procedure, at any time, to customize the security-check configuration. You can also deploy the same descriptor file to other instances of MobileFirst Server on which the same application is registered, or reuse relevant portions of the configuration in other application-descriptor files.

Results

After completing the configuration changes, you can see in the **Security-Check Configurations** table on the application **Security** console page a list of the properties that you configured and their current and default values. In addition, you can see your property configurations in the application descriptor: in the console, go to the application **Configuration Files** tab. In the **Application-Descriptor JSON File** section, you can see a copy of the application-descriptor JSON file. Search for the name of the configured security check within the `securityCheckConfigurations` object. The nested security-check object should contain the names and values of your configured properties. In the following template, replace *SecurityCheckName* with the name of the security check that you configured:

```
"SecurityCheckConfigurations": {
    "SecurityCheckName": {
        "property1Name": "property1Value",
        ["property2Name": "property2Value",
         ...]
      }
    }
}
```

Configuring device single sign-on (SSO):

Enable device single sign-on (SSO) to share the state of a security check among multiple applications on the same device.

About this task

You can enable device single sign-on (SSO) for any custom security check to share the state of this check with other application instances that are running on the same device. For example, you can use device SSO to implement an authentication flow whereby successful user log in from one application is applicable also to other applications on the same device.

Device SSO is configured in the application-descriptor JSON file by using the predefined **enableSSO** security-check configuration property.

Note:

- While device SSO can technically be enabled for any custom security check, ensure that enabling this feature matches the logic of the target security check. Namely, avoid enabling device SSO for security checks that are inherently specific to your application, such as application-authenticity validation.
- Configuration of the device SSO property is done only at the application level. You do not define or configure the **enableSSO** property as part of the implementation of a custom security check.
- Using device SSO might have performance implications.
- The remember-me feature of the UserAuthenticationSecurityCheck base class cannot be used together with a device-SSO configuration.

Procedure

Enable device SSO for a specific security check by using one of the following alternative methods:

- **Using IBM MobileFirst Platform Operations Console** (the console)
  1. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Security** tab.
  2. In the **Security-Check Configurations** section, select **Create New**, or select the edit icon for an existing security-check configuration (if exists).
  3. In the Configure Security-Check Properties dialog window, select the custom security check for which you want to enable device SSO.
  4. Locate the **Enable Device SSO** configuration field, and select `true`. You can also configure other properties of the security check. When you are done, select **OK** to apply your changes.

  You can delete or edit your security-check configuration, including the device-SSO configuration, at any time, by selecting the relevant action icon for your security check in the security-check configurations table.
- **Editing the application-descriptor file**
  1. Create a local copy of the application-descriptor JSON file. See "Application configuration" on page 7-3.
  2. Edit your local copy to enable device SSO for your selected custom security check: device SSO is enabled by setting the `enableSSO` property of a custom security check to `true`. The property configuration is contained within a security-check object that is nested in a `securityCheckConfigurations` object. Locate these objects in your application descriptor file, or create them if they are missing. In the following template, replace *SecurityCheckName* with the name of your selected security check:

```
"securityCheckConfigurations": {
    "SecurityCheckName": {
        [...]
        "enableSSO": true
    }
}
```

For example, the following descriptor-file snippet enables **enableSSO** property for a UserAuthenticationSC security check that also configures other properties:

```
"securityCheckConfigurations": {
    "UserAuthenticationSC": {
        "maxAttempts": "4",
        "failureStateExpirationSec": "120",
        "enableSSO": true
    }
}
```

3. Deploy your copy of the application-descriptor JSON file to MobileFirst Server. See “Application configuration” on page 7-3.

To disable device SSO for your security check, create a new copy of the application-descriptor file, delete the **enableSSO** configuration or set the property value to `false`, and redeploy the descriptor file to the server.

Results

After you successfully enable device SSO for your selected security check, you can see in the **Security-Check Configurations** table on the application **Security** console page, that the value of the **Enable Device SSO** property for your configured security check is `true`. In addition, you can see the device-SSO property definition in the application descriptor: in the console, go to the application **Configuration Files** tab. In the **Application-Descriptor JSON File** section, you can see a copy of the application-descriptor JSON file. Search for the name of the configured security check within the `securityCheckConfigurations` object. The nested security-check object should contain an `"enableSSO": true` entry. In the following template, replace *SecurityCheckName* with the name of the security check that you configured:

```
"securityCheckConfigurations": {
    "SecurityCheckName": {
        [...]
        "enableSSO": true
    }
}
```

To test device SSO, enable this feature for the same security check from multiple applications. Then attempt to access resources that are protected by this security check from multiple applications on the same device. You should be required to pass the security check only once, for the first resource request. For example, for a user-login scenario, after you successfully log in from one application, the log in from the second application on the same device should succeed automatically, without any user input.

Access tokens

Learn more about the access tokens that are generated by the security framework, and how to configure these tokens.

A MobileFirst access token is a digitally signed entity that describes the authorization permissions of a client. After the client's authorization request for a specific scope is granted, and the client is authenticated, the authorization server's token endpoint sends the client an HTTP response that contains the requested access token. For more about the authorization flow and token-generation process, see “End-to-end authorization flow” on page 7-266.

Note: The access token is signed with the MobileFirst Server keystore. For production-level security, configure the server to use your own keystore. See "Configuring the MobileFirst Server keystore" on page 7-316.

Structure of the MobileFirst access token

The MobileFirst access token contains the following information:
- Client ID - a unique identifier of the client.
- Scope - the scope for which the token was granted (see OAuth scopes). This scope does not include the mandatory application scope (see "Configuring a mandatory application scope" on page 7-276).
- Token-expiration time - the time at which the token becomes invalid (expires), in seconds. See "Token expiration."

Token expiration

The granted access token remains valid until its expiration time elapses. The access token's expiration time is set to the shortest expiration time from among the expiration times of all the security checks in the scope. But if the period until the shortest expiration time is longer than the application's maximum token-expiration period, the token's expiration time is set to the current time plus the maximum expiration period. The default maximum token-expiration period (validity duration) is 3,600 seconds (1 hour), but it can be configured by setting the value of the **maxTokenExpiration** application-descriptor property. See "Configuring the maximum access-token expiration period."

Configuring the maximum access-token expiration period

Configure the maximum validity duration (expiration period) of access tokens that are obtained by the application.

About this task

A generated MobileFirst access token has an expiration period, which determines the duration for which the token is valid. See."Token expiration." The maximum access-token expiration period is configured by setting the value of the **maxTokenExpiration** property of the application descriptor. The default value of this property is 3,600 seconds (1 hour). Follow the outlined procedure to configure this property.

Procedure

Configure the application's maximum access-token expiration period by using one of the following alternative methods:
- **Using the IBM MobileFirst Platform Operations Console** (the console)
  1. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Security** tab.
  2. In the **Token Configuration** section, set the value of the **Maximum Token-Expiration Period (seconds)** field to your preferred value, and select **Save** to save the change.
     You can repeat this procedure, at any time, to change the maximum token-expiration period, or select **Restore Default Values** to restore the default value.
- **Editing the application-descriptor file**

1. Create a local copy of the application-descriptor JSON file. See "Application configuration" on page 7-3.
2. Edit your local copy to define a `maxTokenExpiration` key and set its value to the maximum access-token expiration period, in seconds:

   ```
   "maxTokenExpiration": max_token_expiration_period
   ```

   For example, the following code sets the application's maximum token-expiration period to 7,200 seconds (2 hours):

   ```
   {
       ...
       "maxTokenExpiration": 7200
   }
   ```
3. Deploy your copy of the application-descriptor JSON file to MobileFirst Server. See "Application configuration" on page 7-3.

Access-token response

Learn about the structure of a successful response to a client's access-token acquisition request.

Note: The structure of a valid access-token response is relevant if you use the low-level WLAuthorizationManager class and manage the OAuth interaction between the client and the authorization and resource servers yourself, or if you use a confidential client. If you are using the high-level WLResourceRequest class, which encapsulates the OAuth flow for accessing protected resources, the security framework handles the processing of access-token responses for you. See "Client security APIs" and "Confidential clients" on page 7-279.

A successful HTTP response to an access-token request contains a JSON object with the access token and additional data. Following is an example of a valid-token response from the authorization server; (actual access tokens are longer than shown in the example):

```
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",
    "scope": "scopeElement1 scopeElement2"
}
```

The token-response JSON object has these property objects:

- **`token_type`** - the token type is always `"Bearer"`, in accordance with the OAuth 2.0 Bearer Token Usage specification.
- **`expires_in`** - the expiration time of the access token, in seconds.
- **`access_token`** - the generated access token.
- **`scope`** - the requested scope.

The **`expires_in`** and **`scope`** information is also contained in the token (**`access_token`**). See "Structure of the MobileFirst access token" on page 7-304.

Client security APIs

Learn about the MobileFirst security client APIs for issuing resource requests and handling security challenges.

OAuth resource-request APIs

IBM MobileFirst Platform Foundation provides two alternative sets of OAuth client APIs for accessing protected resources:

- The WLResourceRequest class is a high-level API that encapsulates the OAuth flow for accessing a protected resource, and handles the required interaction with the authorization and resource servers. See the documentation of this class for your development platform and programming language.
- The WLAuthorizationManager class is a low-level API for managing the OAuth interaction between the client and the authorization server. In addition, you need to write the code for interacting with the resource server. Sample custom resource-request implementations, which use the WLAuthorizationManager class, are provided to help get your started. See the documentation of this class and the provided sample for your development platform and programming language:
  - "Objective-C custom resource-request implementation sample" on page 7-308
  - "Java custom resource-request implementation sample" on page 7-309
  - "JavaScript custom resource-request implementation sample" on page 7-311
  - "C# custom resource-request implementation sample" on page 7-312

Challenge-handler APIs

The client application uses challenge handlers to handle the client-side security logic and the related user interaction, and respond to security challenges. See "OAuth scopes, security checks, and challenge handlers" on page 7-266. You must implement and register a challenge handler for each custom security check that is applicable to your application (namely, security checks that are used to protect resources that are required by the application). In addition, you can customize the default MobileFirst challenge handler for displaying the user interface (UI) of the mobile-application management features (see "Mobile-application management" on page 10-15).

Creating a challenge handler

When communicating directly with MobileFirst Server, create a MobileFirst security-check challenge handler:

- In iOS Objective C or Swift code, create a class that extends the SecurityCheckChallengeHandler class.
- In Android Java code, create a class that extends the SecurityCheckChallengeHandler class.
- In Windows C# code, create a class that extends the Worklight.SecurityCheckChallengeHandler class.
- In web application or cross-platform (hybrid) Cordova application JavaScript code, call the WL.Client method createSecurityCheckChallengeHandler (which both creates and registers the challenge handler).

In gateway topologies, create a custom gateway challenge handler:

- In iOS Objective C or Swift code, create a class that extends the GatewayChallengeHandler class.
- In Android Java code, create a class that extends the GatewayChallengeHandler class.
- In Windows C# code, create a class that extends the GatewayChallengeHandler class.

- In web application or cross-platform (hybrid) Cordova application JavaScript code, call the WL.Client method createGatewayChallengeHandler (which both creates and registers the challenge handler).

Registering a challenge handler

Use the relevant API to register your challenge handler:

- In iOS Objective C or Swift code, call the WLClientWLClient method registerChallengeHandler.
- In Android Java code, call the WLClient method registerChallengeHandler.
- In Windows C# code, call the WorklightClient method RegisterChallengeHandler. See "C# client-side API for Windows 10 Universal Windows Platform and Windows 8 Universal apps" on page 8-6.
- In web application or cross-platform (hybrid) Cordova application JavaScript code, call the WL.Client method createSecurityCheckChallengeHandler or createGatewayChallengeHandler (which both creates and registers the challenge handler).

The security-challenge object

The security challenge is passed to the application within a JSON object that contains data pairs of a security-check name and an optional JSON object with additional data (or `null` if no additional data is required):

```
{
  "challenges": {
    "SecurityCheck1":null,
    "SecurityCheck2":{
      "PropertyName": "PropertyValue"
    [...]
    }
  }
}
```

Sample implementations and guidelines

You can find sample challenge-handler implementations and related development guidelines in the following IBM MobileFirst Platform Foundation Development Center tutorials. See the relevant tutorial for your development platform.

- The CredentialsValidationSecurityCheck tutorials demonstrate how to implement a challenge handler for the CredentialsValidationSecurityCheck security-check base class (see "The security-check base and sample classes" on page 7-290).
- The UserAuthenticationSecurityCheck tutorials demonstrate how to implement a challenge handler for the UserAuthenticationSecurityCheck security-check base class (see "The security-check base and sample classes" on page 7-290).

Sample custom resource-request implementations using WLAuthorizationManager

Sample custom OAuth resource-request implementations that use the WLAuthorizationManager class.

Select the sample that matches your development platform and programming language.

Objective-C custom resource-request implementation sample:

An Objective-C sample for acquiring data from a protected resource by using the MobileFirst WLAuthorizationManager class.

The sample contains two methods of a class that implements NSURLConnectionDelegate. The sample implements a standard OAuth flow: first, a resource request is sent without an access token. This request is expected to fail with an authorization error. Then, WLAuthorizationManager is used to obtain an access token for the resource's protecting scope, and the request is sent again with the obtained access token as an authorization header. The resource request is created by using a standard NSMutableURLRequest object.

```
/**
 * Sends a request to access the specified protected resource.
 **/
- (void) sendRequest {
    // Set the resource URL
    NSString *resourceString = @"http://localhost:3000/MyResource/MyProtectedProcedure";

    // Create the request to access the resource URL
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:resourceString]];

    if(_accessToken) {
        // Add an access token to the request
        [request addValue:_accessToken.asAuthorizationRequestHeaderField forHTTPHeaderField:@"Authorization"];
    }

    // Create a URL connection that sends the request. The response is returned via the connection:didReceiveResponse delegate.
    NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
}

/**
 * NSURLConnectionDelegate didReceiveResponse method, which implements the MobileFirst OAuth authorization flow.
 **/
-(void) connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {
    WLAuthorizationManager *authorizationManager = [WLAuthorizationManager sharedInstance];

    // Check whether access to the resource requires authorization
    if([authorizationManager isAuthorizationRequiredForResponse:response]) {

        // Get the status from the response
        int status = (int)[(NSHTTPURLResponse *)response statusCode];
        NSString *scope;

        switch (status) {
            case 401: // Invalid-token error (invalid_token)
                // Clear the access token (if exists)
                [authorizationManager clearAccessToken:_accessToken];
                break;
            case 403: // Insufficient-scope error (insufficient_scope)
                // Get the resource scope from response
                scope = [authorizationManager resourceScopeFromResponse:response];
                break;
            case 409: // Server-conflict error
                // Resend the request
                [self sendRequest];
                return;
            default: // Authorization-server error
                NSLog(@"%@",@"Error from the authorization server");
                return;
        }

        // Obtain an access token for the scope from the authorization server
        [authorizationManager obtainAccessTokenForScope:scope withCompletionHandler:^(AccessToken *accessToken, NSError *error) {
            if (!error) {
                // Save the access token
                _accessToken = accessToken;
                [self sendRequest];
            } else {
                // Error obtaining an access token
                NSLog(@"%@",[error localizedDescription]);
            }
        }];
    }
    else { // The resource does not require authorization
        // Initialize an object that will be used to receive the response data from the resource
        _responseData = [[NSMutableData alloc] init];
        NSLog(@"Meta-data response from resource: %@",[response description]);
    }
}
```

Java custom resource-request implementation sample:

This sample demonstrates how to get data from a protected resource by using a custom HttpRequest object and the MobileFirst AuthorizationManager API.

The sample implements a standard OAuth flow: first, a resource request is sent without an access token. This request is expected to fail with an authorization error. Then, WLAuthorizationManager is used to obtain an access token for the resource's protecting scope, and the request is sent again with the obtained access token as an authorization header. The resource request is created by using a standard HttpURLConnection object.

```
package com.sample.oauthdemoandroid;

import android.os.AsyncTask;

import com.worklight.wlclient.api.WLAccessTokenListener;
import com.worklight.wlclient.api.WLAuthorizationManager;
import com.worklight.wlclient.api.WLClient;
import com.worklight.wlclient.api.WLFailResponse;
import com.worklight.wlclient.auth.AccessToken;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;
import java.util.Map;

public class CustomRequestAsyncTask extends AsyncTask<Object, Void, Void> {

    public static final String HEADER_AUTHORIZATION = "Authorization";
    private Object[] params;

    @Override
    protected Void doInBackground(Object[] params) {
        android.os.Debug.waitForDebugger();  // for debugging
        this.params = params;
        sendRequest(null);
        return null;
    }

    private void sendRequest(AccessToken accessToken) {
        HttpURLConnection urlConnection = null;
        try {
            // Create the request to access the resource URL
            URL url = new URL(WLClient.getInstance().getServerUrl().toString() + params[0]);
            urlConnection = (HttpURLConnection) url.openConnection();
            if (accessToken != null) {
                // Add an access token to the request
                urlConnection.setRequestProperty(HEADER_AUTHORIZATION, accessToken.getAsAuthorizationRequestHeader());
            }

            // Send the request
            Map<String, List<String>> headerFields = urlConnection.getHeaderFields();

            // Check whether the request succeeded
            int responseCode = urlConnection.getResponseCode();
            if (200 <= responseCode && responseCode <= 299) {
```

```
            customRequestSuccess(urlConnection);
        } else {
            // Check whether access to the resource requires authorization
            WLAuthorizationManager wlAuthorizationManager = WLAuthorizationManager.getInstance();
            if (wlAuthorizationManager.isAuthorizationRequired(responseCode, headerFields)) {
                switch (responseCode) {
                    case 409: // Server-conflict error
                        // Resend the request
                        sendRequest(accessToken);
                        break;
                    case 401: // Invalid access token, or no access token
                        // Clear the access token (if exists)
                        if (accessToken != null) {
                            wlAuthorizationManager.clearAccessToken(accessToken);
                        }
                        // Obtain a valid access token and resend the request
                        resendWithAccessToken(headerFields);
                        break;
                    case 403: // Insufficient-scope error
                        // Get the resource scope from the response and resend the request
                        resendWithAccessToken(headerFields);
                        break;
                    default: // Unexpected error
                        customRequestFailure(urlConnection);
                }

            } else {
                customRequestFailure(urlConnection);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (urlConnection != null) {
            urlConnection.disconnect();
        }
    }
}

private void customRequestSuccess(HttpURLConnection urlConnection) throws IOException {
    // TODO: Implement the method.
}

private void customRequestFailure(HttpURLConnection urlConnection) throws IOException {
    // TODO: Implement the method.
}

private void resendWithAccessToken(Map<String, List<String>> headerFields) {
    WLAuthorizationManager wlAuthorizationManager = WLAuthorizationManager.getInstance();
    // Get the resource request from the response
    String scope = wlAuthorizationManager.getResourceScope(headerFields);
    // Obtain an access token and resend the request
    CustomRequestObtainAccessTokenListener customRequestObtainAccessTokenListener = new CustomRequestObtainAccessTokenL
    wlAuthorizationManager.obtainAccessToken(scope, customRequestObtainAccessTokenListener);
}

private class CustomRequestObtainAccessTokenListener implements WLAccessTokenListener {

    @Override
    public void onSuccess(AccessToken accessToken) {
        sendRequest(accessToken);
    }

    @Override
    public void onFailure(WLFailResponse response) {
            // TODO: Implement the method.
        }
    }
}
```

JavaScript custom resource-request implementation sample:

A JavaScript sample for acquiring data from a protected resource by using the MobileFirst WLAuthorizationManager class.

The sample implements a standard OAuth flow: first, a resource request is sent without an access token. This request is expected to fail with an authorization error. Then, WLAuthorizationManager is used to obtain an access token for the resource's protecting scope, and the request is sent again with the obtained access token as an authorization header. The resource request is created by using a standard XMLHttpRequest object.

```
function sendCustomRequest() {
    sendRequest('http://localhost:3000/v1/apps/1234/test', null)
        .always(
            function(response) {
                alert(response);
            }
        );
}

/**
 * Sends a request with the provided access token to the specified protected-resource URL.
 **/
function sendRequest(url, accessToken) {
    // Use JavaScript promises for asynchronous operations
    var dfd = WLJQ.Deferred();

    // Create the custom resource request
    var xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);

    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            var status = xhr.status;
            if (status >= 200 && status <= 299) {
                dfd.resolve(xhr.responseText);
            }
            else {
                var headers = xhr.getAllResponseHeaders();

                // Check whether access to the resource requires authorization
                if (WLAuthorizationManager.isAuthorizationRequired(status, headers)) {
                    if (status === 409) { // Server-conflict error
                        // Resend the request
                        sendRequest(url, accessToken)
                            .then(
                                function(response) {
                                    dfd.resolve(response);
                                },
                                function(error) {
                                    dfd.reject(error);
                                }
                            );
                    } else if (status === 401) { // Invalid access token, or no access token
                        // Check whether the access token is invalid
                        if (isInvalidTokenError(xhr)) {
                            // Clear the invalid access token
                            WLAuthorizationManager.clearAccessToken(accessToken).always(
                                function() {
                                    // Obtain a valid access token and resend the request
                                    resendWithAccessToken(null);
                                });
                        } else {
                            // Obtain a valid access token and resend the request
                            resendWithAccessToken(null);
                        }
                    } else { // status = 403 - insufficient-scope error
                        // Get the resource scope from the response
                        var scope = WLAuthorizationManager.getResourceScope(headers);
                        // Obtain an access token for the scope
                        resendWithAccessToken(scope);
                    }
                } else { // Unexpected error
                    dfd.reject("Failure - received response " + xhr.responseText);
                }
            }
        }
    };

    // If an access token was obtained, add the token to the request as an authorization header
    if (accessToken !== null) {
        xhr.setRequestHeader("Authorization", accessToken.asAuthorizationRequestHeader);
    }

    xhr.send();

    return dfd.promise();


    function resendWithAccessToken(scope) {
        WLAuthorizationManager.obtainAccessToken(scope)
            .then(
                function(accessToken) {
                    // The access token was obtained successfully.
                    // Construct the request again, and add the access token as an authorization hea
                    sendRequest(url, accessToken)
                        .then(
                            function(response) {
                                dfd.resolve(response);
                            },
                            function(error) {
                                dfd.reject(error);
                            }
                        );
                },
                function(error) {
                    // Failed to obtain an access token. Reject the request.
                    dfd.reject(error);
                }
            );
    }
}

function isInvalidTokenError(xhr) {
    var authHeader = xhr.getResponseHeader('WWW-Authenticate');
    return (authHeader.indexOf("invalid_token") >= 0);
}
```

C# custom resource-request implementation sample:

This sample demonstrates how to get data from a protected resource by using a HttpWebRequest object and the MobileFirst WLAuthorizationManager API for Windows 8 Universal app.

After a response was first received, the WLAuthorizationManager class 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.

```
class WebReqInfo {
  public HttpWebRequest request = null;
}

private async void externalOAuthFlow() {
  // Create a HttpWebrequest object to the desired URL.
  HttpWebRequest request = (HttpWebRequest)WebRequest.Create("http://localhost:3000/v1/apps/1234/test");
  request.BeginGetResponse(new AsyncCallback(responseCallback), new WebReqInfo(){ request=request });
}

private void responseCallback(IAsyncResult asyncResult) {
  WebReqInfo wReqInfo = null;
  HttpWebResponse response = null;
  try {
    /* Send the request to the server and wait for it to complete */
    wReqInfo = (WebReqInfo)asyncResult.AsyncState;

    // Get response
    response = wReqInfo.request.EndGetResponse(asyncResult) as HttpWebResponse;

    WLResponse newResponse = new WLResponse(response);
    int sCode = newResponse.getStatus().GetHashCode();

    // Check that the response conforms to the MFP protocol
    if (WLAuthorizationManager.getInstance().isAuthorizationRequired(response)) {
      // Get required scope from response
      String scope = WLAuthorizationManager.getInstance().getAuthorizationScope(response);

   //Obtain authorization header for the scope
      WLAuthorizationManager.getInstance().obtainAuthorizationHeader(scope, new MyResponseListener());
    }
    else {
      // At this point a response from the resource was received, process it:
      processResponse(response)
    }
  } catch (Exception e) {
    Debug.WriteLine("Exception during request");
  }
}

public class MyResponseListener : WLResponseListener {
  HttpWebRequest request;
  public MyResponseListener(HttpWebRequest request) {
    this.request = request;
  }
  class WebReqInfo {
    public HttpWebRequest request = null;
  }

  // This method will be invoked if an authorization header was obtained successfully:
  public void onSuccess(WLResponse response) {

    // Add the obtained authorization header to the original request
    WLAuthorizationManager.getInstance().addCachedAuthorizationHeader(request);
    try {
      // Resend the request
      request.BeginGetResponse(new AsyncCallback(resendRequestCallback), new WebReqInfo(){request=request });
    } catch (Exception e) {
      Debug.WriteLine("Unable to obtain token");
    }
  }
  // This method will be invoked if an authorization header was not obtained:
  public void onFailure(WLFailResponse response) {
    Debug.WriteLine("Unable to obtain Token");
  }

  private void resendRequestCallback(IAsyncResult asyncResult) {
    WebReqInfo wReqInfo = null;
    HttpWebResponse response = null;
    try {
      /* Send the request to the server and wait for it to complete */
      wReqInfo = (WebReqInfo)asyncResult.AsyncState;
      response = wReqInfo.request.EndGetResponse(asyncResult) as HttpWebResponse;
      processResponse(response);
    } catch (Exception e) {
      Debug.WriteLine("Exception during request");
    }
  }
}
```

For more information about WLAuthorizationManager, see "C# client-side API for Windows 10 Universal Windows Platform and Windows 8 Universal apps" on page 8-6.

Configuring IBM WebSphere DataPower as the OAuth authorization server

About this task

The MobileFirst security framework is built around an authorization server that implements the OAuth protocol, and exposes the OAuth endpoints with which the client interacts. MobileFirst Server implements custom security logic and advanced security features on top of the authorization server. By default, MobileFirst Server functions also as the OAuth authorization server. However, you can configure IBM WebSphere DataPower (DataPower) to act as the authorization server, and interact with MobileFirst Server. This design provides you with enhanced flexibility in setting up production topologies, for example, deploying the DataPower authorization server in the DMZ.

Note: The basic building blocks of the security framework (security checks and challenge handlers) are unaffected by this mode. The behavior of the building blocks is the same regardless of whether the authorization server is MobileFirst Server or DataPower.

The integration of the MobileFirst security framework with DataPower as the authorization server is achieved by using the provided MobileFirst DataPower pattern file, `dp-external-az-pattern.zip`. You can get this file from the IBM MobileFirst Platform Operations Console: from the console **Dashboard**, select **Download Center**, and then select the **Tools** tab. In the **MobileFirst External Authorization Server Pattern** section of the Tools tab, select **Download** and save the pattern to your preferred location..

To use DataPower as the authorization server, deploy the provided pattern to your DataPower appliance and configure MobileFirst Server to interact with DataPower as the authorization server, as outlined in the following procedure.

Note: When using DataPower as the authorization server, configure client applications to connect to the DataPower appliance instead of connecting directly to MobileFirst Server. For example, in an iOS application, set the **`wlServerHost`** and

wlServerPort properties in mfpclient.plist to the host IP address and port of the DataPower appliance. If you are using a self-signed SSL certificate for DataPower, you also need to import this certificate into the client application.

Procedure

- **Preliminary steps**
  1. Create a key pair for the DataPower SSL certificate with a public key named azserver-sscert.pem and a private key named azserver-privkey.pem. The exact procedure for creating the SSL key pair for production depends on your certificate authority, and therefore cannot be documented here. However, during development you can run the following commands from a command-line terminal to create a self-signed certificate:
     ```
     openssl req -x509 -newkey rsa:4096 -keyout azserver-privkey.pem \
      -out azserver-sscert.pem -days 365 -nodes
     ```
  2. Create a key to be used by DataPower for encryption of OAuth tokens. The key is a hexadecimal string whose size is at least 32 bytes. Save the key to a file named key. Following is a sample key:
     `0xabcd1234abcd1234abcd1234abcd1234abcd1234abcd1234abcd1234abcd1234`
  3. Create a secret file that is named secret. This secret is used by DataPower to authenticate itself with MobileFirst Server. The secret file is a text file that contains a JSON object with a secret entry whose string value defines the secret. The following sample defines a "12345" secret:
     ```
     {"secret":"12345"}
     ```
- **Deploying the MobileFirst DataPower pattern**
  In the WebGUI of your DataPower appliance,
  1. Create a new DataPower domain.
  2. Switch to the new domain, and upload the file that contains the public and private SSL-certification keys that you created in preliminary Step 1 (azserver-sscert.pem and azserver-privkey.pem) to the cert:/// directory.
  3. Upload the key file that you created in preliminary Step 2 to the same directory.
  4. Upload the secret file that you created in preliminary Step 3 to the local:/// directory.
  5. Select **Blueprint Console** in the WebGUI navigation sidebar. Then select the **Patterns** tab, and import the pattern by selecting the graphical import button (next to the **New Pattern** button): .
  6. Select the pattern archive file (dp-external-az-pattern.zip), and wait for the import to complete.
  7. Select **Deploy**, and provide the input in the four required fields: enter a name for the service, the URL of your MobileFirst Server, and the DataPower IP address and port that you want the authorization server to listen to.
  8. After the pattern is deployed, select the **Services** tab to ensure that the new authorization service started successfully.
- **Configuring MobileFirst Server to work with DataPower**
  1. Add the following JNDI entries to the application-server configuration file of your MobileFirst Server (server.xml). Replace the *your_secret* placeholder with the secret that you defined in your secret file in preliminary Step 3 (for example, "12345"). This secret is used for authenticating the DataPower appliance.

```
<jndiEntry jndiName="mfp/mfp.authorization.server" value="external"/>
<jndiEntry jndiName="mfp.external.authorization.server.secret" value="your_secret"/>
<jndiEntry jndiName="mfp.external.authorization.server.introspection.url"
         value="https://<DataPower host address>:8443/az/v1/introspection"/>
```

2. Import the DataPower public-key certificate file (`azserver-sscert.pem`) into the WebSphere Application Server Liberty keystore of your MobileFirst Server to allow the server to connect over SSL. You can run the following commands to import the key into the keystore:

   ```
   openssl x509 -outform der -in azserver-sscert.pem -out azserver-sscert.der
   keytool -import -keystore key.jks -file azserver-sscert.der
   ```

   The first command takes the certificate and converts it into DER format. The second command uses the Java **`keytool`** utility to import the certificate into the WebSphere Application Server Liberty keystore of your MobileFirst Server.
3. Create a special empty scope-element mapping to be used by DataPower. You need to create this mapping in every application that is registered with MobileFirst Server. To create a mapping, select the **Security** tab on the application page, and then select **Create New** under **Security-Elements Mapping**. In the Add New Scope-Element Mapping dialog window, create a new mapping (for example, a mapping named `none`), and leave the **Scope Element** field empty (do not map the scope to any security check). Select **Add** to complete the mapping.

- **Protecting resources with an empty scope**

  To protect a resource by requiring an access token, you need to explicitly set the resource's protecting scope to the empty scope element that you mapped in Step 3 of the server-configuration procedure. For information about how to protect your resources with a scope, see “OAuth resource protection” on page 7-271. On the client side, call WLResourceRequest with a scope parameter that is set to the same empty scope element.

Configuring the MobileFirst Server keystore

Configure MobileFirst Server to use your own keystore for production-level security.

About this task

A keystore is a repository of security keys and certificates that is used to verify and authenticate the validity of parties involved in a network transaction. The MobileFirst Server keystore defines the identity of MobileFirst Server instances, and is used to digitally sign OAuth tokens and Direct Update packages. In addition, when a MobileFirst adapter communicates with a back-end server using mutual HTTPS (SSL) authentication, the keystore is used to validate the SSL-client identity of the MobileFirst Server instance.

For production-level security, during the move from development to production the administrator must configure MobileFirst Server to use a user-defined keystore. The default MobileFirst Server keystore is intended to be used only during development.

Note:

- To use the keystore to verify the authenticity of a Direct Update package, statically bind the application with the public key of the MobileFirst Server identity that is defined in the keystore. See “Implementing secure Direct Update on the client side” on page 7-239.

- Reconfiguring the MobileFirst Server keystore after production should be considered carefully. Changing the configuration has the following potential effects:
  - The client might need to acquire a new OAuth token in place of a token signed with the previous keystore. In most cases, this process is transparent to the application.
  - If the client application is bound to a public key that does not match the MobileFirst Server identity in the new keystore configuration, Direct Update fails. To continue getting updates, bind the application with the new public key, and republish the application. Alternatively, change the keystore configuration again to match the public key to which the application is bound. See "Implementing secure Direct Update on the client side" on page 7-239.
  - For mutual SSL authentication, if the SSL-client identity alias and password that are configured in the adapter are not found in the new keystore, or do not match the SSL certifications, SSL authentication fails. See the adapter configuration information in Step 2 of the following procedure.

Follow these steps to configure MobileFirst Server to use your own keystore:

Procedure

1. Create a Java keystore (JKS) or PKCS 12 keystore file with an alias that contains a key pair that defines the identity of your MobileFirst Server. If you already have an appropriate keystore file, skip to the next step.

   Note: The type of the alias key-pair algorithm must be RSA. The following instructions explain how to set the algorithm type to RSA when using the **keytool** utility.

   You can use a third-party tool to create the keystore file. For example, you can generate a JKS keystore file by running the Java **keytool** utility with the following command (where <*keystore name*> is the name of your keystore and <*alias name*> is your selected alias):

   ```
   keytool -keystore <keystore name> -genkey -alias <alias name> -keylag RSA
   ```

   The following sample command generates a `my_company.keystore` JKS file with a **`my_alias`** alias:

   ```
   keytool -keystore my_company.keystore -genkey -alias my_alias -keyalg RSA
   ```

   The utility prompts you to provide different input parameters, including the passwords for your keystore file and alias.

   Note: You must set the `-keyalg RSA` option to set the type of the generated key algorithm to RSA instead of the default DSA.

   To use the keystore for mutual SSL authentication between a MobileFirst adapter and a back-end server, also add a MobileFirst SSL-client identity alias to the keystore. You can do this by using the same method that you used to create the keystore file with the MobileFirst Server identity alias, but provide instead the alias and password for the SSL-client identity.
2. Configure MobileFirst Server to use your keystore: in the IBM MobileFirst Platform Operations Console navigation sidebar, select **Runtime Settings**, and then select the **Keystore** tab. Follow the instructions on this tab to configure your user-defined MobileFirst Server keystore. The steps include uploading your keystore file, indicating its type, and providing your keystore password, the name of your MobileFirst Server identity alias, and the alias password.

When configured successfully, the Status changes to "User Defined". Otherwise, an error is displayed and the status remains "Default".

The SSL-client identity alias (if used) and its password are configured in the descriptor file of the relevant adapter, within the <sslCertificateAlias> and <sslCertificatePassword> subelements of the <connectionPolicy> element. See “HTTP adapter connectionPolicy element” on page 7-209.

API reference

To develop your native or hybrid applications, refer to the MobileFirst API in JavaScript, Java Platform, Java for Android, and Objective-C for iOS.

Use the MobileFirst API to develop your applications in JavaScript, Java Platform, Java for Android, and Objective-C for iOS.

MobileFirst client-side API

This collection of topics documents the application programming interface (API) for each IBM MobileFirst Platform Foundation client platform.

JavaScript client-side API

You can use JavaScript API to develop apps for all environments.

You can find the description of the API in the following file: JavaScript client-side API.

The other topics in this section contain additional information that you need to use this API to full advantage.

The `options` Object

The `options` object contains properties that are common to all methods. It is used in all asynchronous calls to the IBM MobileFirst Platform Server

Pass an options object for all asynchronous calls to MobileFirst Server. The `options` object contains properties that are common to all methods. Sometimes it is augmented by properties that are only applicable to specific methods. These additional properties are detailed as part of the description of the specific methods.

The common properties of the `options` object are as follows:

```
options = {
  onSuccess: success-handler-function(response),
  onFailure: failure-handler-function(response),
  invocationContext: invocation-context
};
```

The meaning of each property is as follows:

Table 8-1. Options object properties

Property	Description
`onSuccess`	Optional. The function to be invoked on successful completion of the asynchronous call. The syntax of the `onSuccess` function is: `success-handler-`*`function`*`(`*`response`*`)` where *response* is an object that contains at a minimum the following property: **`invocationContext`** The `invocationContext` object that was originally passed to the MobileFirst Server in the options object, or `undefined` if no `invocationContext` object was passed. **`status`** The HTTP response status **Note:** For methods for which the *response* object contains additional properties, these properties are detailed as part of the description of the specific method.

Table 8-1. Options object properties (continued)

Property	Description
onFailure	Optional. The function to be invoked when the asynchronous call fails. Such failures include both server-side errors, and client-side errors that occurred during asynchronous calls, such as server connection failure or timed out calls. **Note:** The function is not called for client-side errors that stop the execution by throwing an exception. The syntax of the onFailure function is: failure-handler-*function*(*response*) where *response* is an object that contains the following properties: **invocationContext** The invocationContext object that was originally passed to the MobileFirst Server in the options object, or undefined if no invocationContext object was passed. **errorCode** An error code string. All error codes that can be returned are defined as constants in the WL.ErrorCode object in the worklight.js file. **errorMsg** An error message that is provided by the MobileFirst Server. This message is for the developer's use only, and should not be displayed to the user. It will not be translated to the user's language. **status** The HTTP response status **Note:** For methods for which the *response* object contains additional properties, these properties are detailed as part of the description of the specific method.

Table 8-1. Options object properties (continued)

Property	Description
invocationContext	Optional. An object that is returned to the success and failure handlers. The `invocationContext` object is used to preserve the context of the calling asynchronous service upon returning from the service. For example, the `invokeProcedure` method might be called successively, using the same success handler. The success handler needs to be able to identify which call to `invokeProcedure` is being handled. One solution is to implement the `invocationContext` object as an integer, and increment its value by one for each call of `invokeProcedure`. When it invokes the success handler, the MobileFirst framework passes to it the `invocationContext` object of the `options` object associated with the invokeProcedure method. The value of the `invocationContext` object can be used to identify the call to `invokeProcedure` with which the results that are being handled are associated.

The WL.ClientMessages object

You can see a list of the system messages that are stored in the `WL.ClientMessages` object, and enable the translation of these system messages.

The `WL.ClientMessages` object is an object that stores the system messages that are defined in the `worklight/messages/messages.json` file. This file is in the `environment` folder of the projects that you generated with IBM MobileFirst Platform Foundation. To enable the translation of a system message, you must override the value of this message in the `WL.ClientMessages` object, as indicated in the following code example:

```
WL.ClientMessages.invalidUsernamePassword="The custom user name and password are not valid";
```

Note: You must override the system messages on a global JavaScript level because some parts of the code run only after the application successfully initialized.

JavaScript client-side push API

You can use JavaScript push API to run push functions in the client-side applications.

You can find the description of the API in the following file: JavaScript client-side push API.

JavaScript web analytics client-side API

You can use the web JavaScript API to develop MobileFirst analytics for your web apps. This API is written in pure JavaScript and is not dependent on any other platform SDKs.

You can find the description of the API in the following file: JavaScript web analytics client-side API.

Note: This SDK must be installed together with the MobileFirst web SDK. See "Developing web applications" on page 7-73.

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. The instructions for the configuration and setup for both types of Xcode projects are provided.

For more information, see "Developing MobileFirst applications" on page 7-24.

You can find the description of the API in the following file: Objective-C client-side API for iOS apps.

Objective-C client-side push API for iOS apps

Use this push API to develop apps for the iOS environment.

Use the Objective-C client-side push API for iOS apps that IBM MobileFirst Platform Foundation provides if you want to access MobileFirst services from iOs applications. For more information about this API, expand the entry for this topic in the **Contents** panel and see the **Overview** topic.

You can find the description of the API in the following file: Objective-C client-side push API for iOS apps.

Objective-C client-side API for hybrid apps

Use this API to develop hybrid apps for iOS environment.

You can use the WL class to handle the initialization of your MobileFirst hybrid application and extend WLAppDelegate class to use the MobileFirst framework API.

You can find the description of the API in the following file: Objective-C client-side API for hybrid apps.

Java client-side API for Android apps

You can use Java API to develop apps for the Android environment.

Use the Java client-side API for Android apps that IBM MobileFirst Platform Foundation provides if you want to access MobileFirst services from Android mobile applications.

You can use this API to develop native apps. If you develop hybrid apps, you can also use the relevant part of this API, either directly by using the WL.NativePage API or by using Cordova.

You can find the description of the API in the following file: Java client-side API for Android apps.

Java client-side push API for Android apps

You can use Java push API to develop apps for the Android environment.

Use the Java client-side push API for Android apps that IBM MobileFirst Platform Foundation provides if you want to access MobileFirst services from Android mobile applications.

You can find the description of the API in the following file: Java client-side push API for Android apps.

C# client-side API for Windows 10 Universal Windows Platform and Windows 8 Universal apps

You can use C# API to develop apps for the Windows 10 Universal Windows Platform (UWP) and Windows 8 Universal environment.

Use this C# client-side API to access MobileFirst services from Windows 10 UWP and Windows 8 Universal applications.

You can find the description of this API in the following file: C# client-side API for Windows 10 Universal and Windows 8 Universal apps.

C# client-side push API for Windows 10 Universal Windows Platform and Windows 8 Universal apps

You can use C# push APIs to develop apps which can send push notifications, for the Windows 10 Universal Windows Platform (UWP) and Windows 8 Universal environment.

Use this C# client-side push API to access MobileFirst push services from Windows 10 UWP and Windows 8 Universal applications.

You can find the description of this API in the following file: C# client-side push API for Windows 10 Universal and Windows 8 Universal apps.

MobileFirst server-side API

Use the server-side API that IBM MobileFirst Platform Foundation 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.

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 provides adapters for SOAP or XML over HTTP, JDBC, and JMS. For more information about developing adapters, see “Developing the server side of a MobileFirst application” on page 7-187.

Security libraries

You can use the security libraries to develop security checks and adapters that use the security context. The libraries provide the required interfaces for adapters and security checks to gain access to the security context. For more information about the MobileFirst security framework, see "MobileFirst security framework" on page 7-265.

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 Java Token Validator API

Use the Java Token Validator API of the MobileFirst Java Token Validator library (`mfp-java-token-validator-8.0.0.jar`) to protect external Java resources by validating the access tokens for these resources.

Package com.ibm.mfp.java.token.validator

This package includes classes for using the introspection endpoint of the security framework's authorization server to validate authorization headers. You can use this library to validate tokens that are used to access resources on an external Java server. See "MobileFirst Java Token Validator" on page 7-274.

You can find the description of the API in the following file: Java Token Validator API.

REST API for the MobileFirst Server administration service

The REST API provides several services to administer runtime adapters, applications, devices, audit, transactions, security, and push notifications.

The REST service API for adapters and applications for each runtime is in `/management-apis/2.0/runtimes/`*`runtime-name`*`/`, where *runtime-name* is the name of the runtime 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/2.0/runtimes/`*`runtime-name`*`/Adapters (POST)` refers to the service for deploying an adapter.

Adapter (GET)

Retrieves metadata of a specific adapter.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**

- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*/adapters/*adapter-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/adapters/myadapter?locale=de_
```

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",
  "displayName" : "MyApplication",
  "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications
  "name" : "SampleAdapter",
  "productVersion" : "8.0",
  "project" : {
    "name" : "myproject",
  },
  "resourceName" : "abc",
  "resourceType" : "APP_DESCRIPTOR",
  "runtimeInfo" : {
    "descriptorXML" : "",
    "resources" : {
      "basePath" : "/mfp/api/adapters/demoAdapter",
      "info" : {
        "description" : "The adapter for bank-end service",
        "title" : "demoAdapter",
      },
      "paths" : {
      },
      "swagger" : "2.0",
    },
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter
  deployTime="2014-04-13T00:18:36.979Z"
  displayName="MyApplication"
  link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications/
  name="SampleAdapter"
  productVersion="8.0"
  resourceName="abc"
  resourceType="APP_DESCRIPTOR">
  <project name="myproject"/>
  <runtimeInfo descriptorXML="">
    <resources
      basePath="/mfp/api/adapters/demoAdapter"
      swagger="2.0">
      <info
        description="The adapter for bank-end service"
        title="demoAdapter"/>
      <paths/>
    </resources>
  </runtimeInfo>
</adapter>
```

Response Properties

The response has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
The optional display name of the artifact.

link
The URL to access detailed information about the deployed artifacts such as application, adapter etc.

name
The name of the adatper

productVersion
The exact product version.

project
The project the artifact belong to.

resourceName
The name of the artifact.

resourceType
The type of the artifact.

runtimeInfo
The runtime information of the adapter

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

The *runtime-info* has the following properties:

descriptorXML
The adapter description in XML

resources
Adapter resource information

The *resource-info* has the following properties:

basePath
The base api path to the adatper

info
The information about the adapter

paths
The adapter methods

swagger
The Swagger version

The *adapter-info* has the following properties:

description
The description of the adapter

title
The title of the adapter

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 (DELETE)

Deletes a specific adapter.

Description

This transaction can run synchronously or asynchronously. If the transaction is 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:
- **mfpadmin**
- **mfpdeployer**

Method

DELETE

Path

/management-apis/2.0/runtimes/*runtime-name*/adapters/*adapter-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/adapters/myadapter?async=f
```

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. The allowed values are `true` and `false`. The default mode 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" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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="8.0">
  <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
      name="myname"
      type="mytype"/>
    <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:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 (POST)

Deploys an adapter.

Description

Deploys an adapter.

The transaction first checks whether the input deployable is valid. Then, it transfers the deployable to the database and to the runtime.

This transaction can run synchronously or asynchronously. If the transaction is 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:
- **`mfpadmin`**
- **`mfpdeployer`**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/adapters

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/adapters?async=false&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.

`async`
Whether the transaction is processed synchronously or asynchronously. The allowed values are `true` and `false`. The default mode 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 Adatper.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-adapter-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <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 Adatper that is deployed.

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_ARTIFACT`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 deployable 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:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/adapters

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/adapters?bookmark=ABC&incl
```

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.

`include`
To show runtimeInfo as part of each adapter.

`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: displayName, deployTime. The default sort mode is: displayName.

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",
      "displayName" : "MyApplication",
      "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applica
      "project" : {
        "name" : "myproject",
      },
      "resourceName" : "abc",
      "resourceType" : "APP_DESCRIPTOR",
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "8.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<adapters
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="8.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deployTime="2014-04-13T00:18:36.979Z"
      displayName="MyApplication"
      link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/application
      resourceName="abc"
      resourceType="APP_DESCRIPTOR">
      <project name="myproject"/>
    </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 one page of adapters is returned.

pageNumber
The page index if only one page of adapters is returned.

pageSize
The page size if only one page of adapters is returned.

prevPageBookmark
The bookmark of the previous page if only one page of adapters is returned.

productVersion
The exact product version.

startIndex
The start index in the total list if only one page of adapters is returned.

totalListSize
The total number of adapters.

The *configlink* has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
The optional display name of the artifact.

link
The URL to access detailed information about the deployed artifacts such as application, adapter etc.

project
The project the artifact belong to.

resourceName
The name of the artifact.

resourceType
The type of the artifact.

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.`

Adapter Configuration (GET)

Retrieves the user configuration of a specific adapter.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/adapters/*adapter-name*/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/adapters/myadapter/config?fla
```

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.

`flattened`
If the parameter is set to `true` (default value), the configuration is a flat list of properties. If the parameter is set to `false`, the configuration is a hierarchy of objects.

`locale`
The locale used for error messages.

`mode`
If no mode is specified, the transaction returns the current user configuration. If the mode `defaults` is specified, the transaction returns the default configuration.

Produces

application/json, application/xml, text/xml

Response

The user configuration of the specified adapter.

JSON Example

```
{
  "adapter" : "myAdapter",
  "connectivity" : {
    "http" : {
      "connectionTimeoutInMilliseconds" : 30000,
      "cookiePolicy" : "BEST_MATCH",
      "dtdvalidationEnabled" : false,
      "maxConcurrentConnectionsPerNode" : 49,
      "maxRedirects" : 11,
      "port" : 444,
      "protocol" : "http",
      "socketTimeoutInMilliseconds" : 30002,
    },
  },
  "properties" : {
    "database" : "test-db",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<adapterconfig adapter="myAdapter">
  <connectivity>
    <http
      connectionTimeoutInMilliseconds="30000"
      cookiePolicy="BEST_MATCH"
      dtdvalidationEnabled="false"
      maxConcurrentConnectionsPerNode="49"
      maxRedirects="11"
      port="444"
      protocol="http"
      socketTimeoutInMilliseconds="30002"/>
  </connectivity>
  <properties database="test-db"/>
</adapterconfig>
```

Response Properties

The response has the following properties:

adapter
The name of the adapter.

connectivity
The connectivity details

properties
The properties of the adapter, mainly for Java adapters

The *connectivity* has the following properties:

http
The HTTP connection details

The *httpdetails* has the following properties:

connectionTimeoutInMilliseconds
The connection timeout value in milliseconds

cookiePolicy
The cookie policy to be used

dtdvalidationEnabled
Whether DTD validation is enabled for syntax and structure of the XML DTD

maxConcurrentConnectionsPerNode
The maximum number of concurrent connections allowed per node

maxRedirects
The maximum number of redirections allowed

port
The port used for the connection

protocol
The HTTP protocol used for connection

socketTimeoutInMilliseconds
The timeout value for socket

The *properties* has the following properties:

database
The name of the datbase to connect to.

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 configuration (PUT)

Sets the user configuration of a specific adapter.

Description

This transaction can run synchronously or asynchronously. If the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/adapters/*adapter-name*/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/adapters/myadapter/config?
```

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. The allowed values are `true` and `false`. The default mode 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

```
{
  "adapter" : "myAdapter",
  "connectivity" : {
    "http" : {
      "connectionTimeoutInMilliseconds" : 30000,
      "cookiePolicy" : "BEST_MATCH",
      "dtdvalidationEnabled" : false,
      "maxConcurrentConnectionsPerNode" : 49,
      "maxRedirects" : 11,
      "port" : 444,
      "protocol" : "http",
      "socketTimeoutInMilliseconds" : 30002,
    },
  },
  "properties" : {
    "database" : "test-db",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<adapterconfig adapter="myAdapter">
  <connectivity>
    <http
      connectionTimeoutInMilliseconds="30000"
      cookiePolicy="BEST_MATCH"
      dtdvalidationEnabled="false"
      maxConcurrentConnectionsPerNode="49"
      maxRedirects="11"
      port="444"
      protocol="http"
      socketTimeoutInMilliseconds="30002"/>
  </connectivity>
  <properties database="test-db"/>
</adapterconfig>
```

Payload Properties

The payload has the following properties:

adapter
The name of the adapter.

connectivity
The connectivity details

properties
The properties of the adapter, mainly for Java adaers

The *connectivity* has the following properties:

http
The HTTP connection details

The *httpdetails* has the following properties:

connectionTimeoutInMilliseconds
The connection timeout value in milliseconds

cookiePolicy
The cookie policy to be used

dtdvalidationEnabled
Whether DTD validation is enabled for syntax and structure of the XML DTD

maxConcurrentConnectionsPerNode
The maximum number of concurrent connections allowed per node

maxRedirects
The maximum number of redirections allowed

port
The port used for the connection

protocol
The HTTP protocol used for connection

socketTimeoutInMilliseconds
The timeout value for socket

The *properties* has the following properties:

database
The name of the datbase to connect to.

Response

The user configuration of the specified adapter.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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-adapterconfig-result
  ok="false"
  productVersion="8.0">
  <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
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-adapterconfig-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:

contentNames
: The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
: The optional file name of the artifact.

name
: The name of the artifact.

type
: The type of the artifact.

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 adapter is not found.
```

500

```
An internal error occurred.
```

Application Authenticity (DELETE)

Deletes specific application authenticity data.

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:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/ *application-env*/*application-version*/authenticity

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplicatior
```

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. The allowed values are `true` and `false`. By default, transactions are processed synchronously.

`locale`
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

Deletes application authenticity.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_APPAUTH",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-appversion-authenticitydata-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="DELETE_APPAUTH"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</delete-appversion-authenticitydata-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
Deletes the app authenticity data of an 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_APPAUTH`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 version is not found.`

500

`An internal error occurred.`

Application Configuration (GET)

Retrieves the configuration of a specific application version.

Roles

Users in the following roles are authorized to perform this operation:
- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/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.

application-env
The application environment.

application-version
The application version number.

Query Parameters

Query parameters are optional.

flattened
If this parameter is set to `true` (which is the default value), the configuration is returned as a flat list of properties. Otherwise, it is returned as a hierarchy of objects.

locale
The locale used for error messages.

mode
If no mode is specified, the method returns the current user configuration. If the `defaults` mode is specified, the method returns the default configuration.

Produces

application/json, application/xml, text/xml

Response

The configuration of the specified application version.

JSON Example

```
{
  "applicationAccessConfig" : {
    "action" : "BLOCKED",
    "downloadLink" : "www.ynet.co.il",
    "message" : "The application is blocked.",
    "multiLanguageMessages" : [
      {
        "locale" : "de",
        "message" : "Bitte updaten!",
      },
      ...
    ],
  },
  "clientLogProfiles" : {
    "level" : "INFO",
    "name" : "com.acme.sub1",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<appconfig>
  <applicationAccessConfig
    action="BLOCKED"
    downloadLink="www.ynet.co.il"
    message="The application is blocked.">
    <multiLanguageMessages>
      <multiLanguageMessage
        locale="de"
        message="Bitte updaten!"/>
      ...
    </multiLanguageMessages>
  </applicationAccessConfig>
  <clientLogProfiles
    level="INFO"
    name="com.acme.sub1"/>
</appconfig>
```

Response Properties

The response has the following properties:

applicationAccessConfig
The access configuration of an application version.

clientLogProfiles
The log filters to collect application logs from devices according to a profile.

The *accessConfig* has the following properties:

action
Application access status

downloadLink
The URL for the new version of the application to download.

message
The message that the user receives when opening the application.

multiLanguageMessages
The notification text in different languages.

The *languages* has the following properties:

locale
The locale for the language

message
The message in the locale

The *logfilters* has the following properties:

level
The severity level. Errors are returned from this level upwards.

name
The logical package name that is used to identify the logger in the mobile application

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or the application version is not found.`

500

`An internal error occurred.`

Application Configuration (PUT)

Sets the configuration of a specific application 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 by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpoperator`**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplicatior
```

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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

`locale`
The locale used for error messages.

Consumes

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "applicationAccessConfig" : {
    "action" : "BLOCKED",
    "downloadLink" : "www.ynet.co.il",
    "message" : "The application is blocked.",
    "multiLanguageMessages" : [
      {
        "locale" : "de",
        "message" : "Bitte updaten!",
      },
      ...
    ],
  },
  "clientLogProfiles" : {
    "level" : "INFO",
    "name" : "com.acme.sub1",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<appconfig>
  <applicationAccessConfig
    action="BLOCKED"
    downloadLink="www.ynet.co.il"
    message="The application is blocked.">
    <multiLanguageMessages>
      <multiLanguageMessage
        locale="de"
        message="Bitte updaten!"/>
      ...
    </multiLanguageMessages>
  </applicationAccessConfig>
  <clientLogProfiles
    level="INFO"
    name="com.acme.sub1"/>
</appconfig>
```

Payload Properties

The payload has the following properties:

applicationAccessConfig
The access configuration of an application version.

clientLogProfiles
The log filters to collect application logs from devices according to a profile.

The *accessConfig* has the following properties:

action
Application access status

downloadLink
The URL for the new version of the application to download.

message
The message that the user receives when opening the application.

multiLanguageMessages
The notification text in differenet languages.

The *languages* has the following properties:

locale
The locale for the language

message
The message in the locale

The *logfilters* has the following properties:

level
The severity level. Errors are returned from this level upwards.

name
The logical package name that is used to identify the logger in the mobile application

Response

The configuration of the specified application version.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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-appconfig-result
  ok="false"
  productVersion="8.0">
  <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
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-appconfig-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:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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.`

Application Descriptor (GET)

Retrieves the application descriptor of a specific application version.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*/descriptor

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/ar
```

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 application descriptor of the specified application version.

JSON Example

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "displayName" : "MyApplication",
  "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications
  "productVersion" : "8.0",
  "project" : {
    "name" : "myproject",
  },
  "resourceName" : "abc",
  "resourceType" : "APP_DESCRIPTOR",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<appdescriptor
  deployTime="2014-04-13T00:18:36.979Z"
  displayName="MyApplication"
  link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications/{ap
  productVersion="8.0"
  resourceName="abc"
  resourceType="APP_DESCRIPTOR">
  <project name="myproject"/>
</appdescriptor>
```

Response Properties

The response has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
The optional display name of the artifact.

link
The URL to access detailed information about the deployed artifacts such as application, adapter etc.

productVersion
The exact product version.

project
The project the artifact belong to.

resourceName
The name of the artifact.

resourceType
The type of the artifact.

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 Environment (GET)

Retrieves the metadata of a specific application environment.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/ap
```

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.

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 environment.

JSON Example

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "displayName" : "MyApplication",
  "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications
  "productVersion" : "8.0",
  "project" : {
    "name" : "myproject",
  },
  "resourceName" : "abc",
  "resourceType" : "APP_DESCRIPTOR",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<appenv
  deployTime="2014-04-13T00:18:36.979Z"
  displayName="MyApplication"
  link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications/{ap
  productVersion="8.0"
  resourceName="abc"
  resourceType="APP_DESCRIPTOR">
  <project name="myproject"/>
</appenv>
```

Response Properties

The response has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
: The optional display name of the artifact.

link
: The URL to access detailed information about the deployed artifacts such as application, adapter etc.

productVersion
: The exact product version.

project
: The project the artifact belong to.

resourceName
: The name of the artifact.

resourceType
: The type of the artifact.

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 environment is not found.`

500

`An internal error occurred.`

Application (GET)

Retrieves the metadata of a specific application.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplicatior
```

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

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "displayName" : "MyApplication",
  "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications
  "productVersion" : "8.0",
  "project" : {
    "name" : "myproject",
  },
  "resourceName" : "abc",
  "resourceType" : "APP_DESCRIPTOR",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<application
  deployTime="2014-04-13T00:18:36.979Z"
  displayName="MyApplication"
  link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications/{a
  productVersion="8.0"
  resourceName="abc"
  resourceType="APP_DESCRIPTOR">
  <project name="myproject"/>
</application>
```

Response Properties

The response has the following properties:

deployTime
: The date in ISO 8601 format when the artifact was deployed.

displayName
: The optional display name of the artifact.

link
: The URL to access detailed information about the deployed artifacts such as application, adapter etc.

`productVersion`
The exact product version.

`project`
The project the artifact belong to.

`resourceName`
The name of the artifact.

`resourceType`
The type of the artifact.

The *project* has the following properties:

`name`
The name of the project, which is the context root of the runtime.

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or the application is not found.`

500

`An internal error occurred.`

Application (POST)

Deploys an application.

Description

A deployable application

It first checks whether the input deployable is valid. Then, it transfers the deployable 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 by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **`mfpadmin`**
- **`mfpdeployer`**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications?async=false&loca
```

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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

locale
The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "applicationKey" : {
    "bundleId" : "com.example",
    "clientPlatform" : "B2C",
    "version" : "com.package.id",
  },
  "displayName" : "Display Name",
  "mandatoryScope" : "appAuthenticity",
  "scopeTokenMapping" : {
    "get-enroll-state" : "",
    "set-enroll-state" : "usernamePassword",
  },
  "securityCheckConfigurations" : {
    "appAuthenticity" : {
      "expirationSec" : "1200",
    },
  },
  "usernamePassword" : {
    "inactivityTimeoutSec" : 30,
    "maxAttempts" : 3,
  },
}
```

Payload Properties

The payload has the following properties:

applicationKey
application key

displayName
: Display Name

mandatoryScope
: scope

scopeTokenMapping
: scope token mapping

securityCheckConfigurations
: security check configuration

usernamePassword
: username password

The *applicationKey* has the following properties:

bundleId
: this is for iOS. Note: For Android, use packageName. For Windows, use assemblyName.

clientPlatform
: License category

version
: version

The *scopeTokenMapping* has the following properties:

get-enroll-state
: get enroll state

set-enroll-state
: set-enroll-state

The *securityCheckConfigurations* has the following properties:

appAuthenticity
: app auth configurations

The *expirationSec* has the following properties:

expirationSec
: expiration in sec

The *usernamePassword* has the following properties:

inactivityTimeoutSec
: inactive time out

maxAttempts
: maximum attemts

Response

The metadata of the deployable.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-application-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <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 deployable.

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_ARTIFACT.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 deployable 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:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/applications

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications?bookmark=ABC&exp
```

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.

expand
Whether an expanded version of the result should be shown. If this parameter is set to false, only a flat list of applications are returned. If the parameter is set to true, the entire hierarchy of environment and versions is returned, too.

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: displayName, deployTime. The default sort mode is: displayName.

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" : [
    {
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "displayName" : "MyApplication",
      "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/appli
      "project" : {
        "name" : "myproject",
      },
      "resourceName" : "abc",
      "resourceType" : "APP_DESCRIPTOR",
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "8.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<applications
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="8.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deployTime="2014-04-13T00:18:36.979Z"
      displayName="MyApplication"
      link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applicati
      resourceName="abc"
      resourceType="APP_DESCRIPTOR">
      <project name="myproject"/>
    </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 one page of applications is returned.

pageNumber
The page index if only one page of applications is returned.

pageSize
The page size if only one page of applications is returned.

prevPageBookmark
The bookmark of the previous page if only one page of applications is returned.

productVersion
The exact product version.

startIndex
The start index in the total list if only one page of applications is returned.

totalListSize
The total number of applications.

The *configlink* has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
The optional display name of the artifact.

link
The URL to access detailed information about the deployed artifacts such as application, adapter etc.

project
The project the artifact belong to.

resourceName
The name of the artifact.

resourceType
The type of the artifact.

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.`

Application License Configuration (POST)

Deploys a license configuration for an application.

Description

A license configuration for an application

The method first checks whether the input deployable is valid. Then, it transfers the deployable 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 by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/license

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/license?async=false&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.

`async`
: Whether the transaction is processed synchronously or asynchronously. Allowed values: `true` and `false`. By default, transactions are processed in synchronous mode.

`locale`
: The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "appID" : "com.package.id",
  "category" : "B2C",
  "licenseType" : "APPLICATION",
}
```

Payload Properties

The payload has the following properties:

appID
: The package name (Android) bundleId (iOS), or package identity (Windows) name of the application

category
: License category

licenseType
: The type of license

Response

The metadata of the deployable.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-app-licenseconfig-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-app-licenseconfig-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 deployable.

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_ARTIFACT`.

`userName`
The user that initiated the transaction.

The *description* has the following properties:

`contentNames`
The optional names of the contained artifacts if multiple artifacts were deployed at once.

`filename`
The optional file name of the artifact.

`name`
The name of the artifact.

`type`
The type of the artifact.

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 deployable 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.`

Application license configuration (GET)

Retrieves the metadata of a specific license configuration for the application.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/license/*application-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/license/myapplication?loca
```

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 license configuration as JSON code.

JSON Example

```
{
  "appID" : "com.package.id",
  "category" : "B2C",
  "licenseType" : "APPLICATION",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<runtime
  appID="com.package.id"
  category="B2C"
  licenseType="APPLICATION"/>
```

Response Properties

The response has the following properties:

appID
: The package name (Android) bundleId (iOS), or package identity (Windows) name of the application

category
: License category

licenseType
: The type of license

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.`

Application Version (GET)

Retrieves the metadata of a specific application version.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/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.

`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 metadata of the specified application version.

JSON Example

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "displayName" : "MyApplication",
  "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applicati
  "productVersion" : "8.0",
  "project" : {
    "name" : "myproject",
  },
  "resourceName" : "abc",
  "resourceType" : "APP_DESCRIPTOR",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<appversion
  deployTime="2014-04-13T00:18:36.979Z"
  displayName="MyApplication"
  link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications/
  productVersion="8.0"
  resourceName="abc"
  resourceType="APP_DESCRIPTOR">
  <project name="myproject"/>
</appversion>
```

Response Properties

The response has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
The optional display name of the artifact.

link
The URL to access detailed information about the deployed artifacts such as application, adapter etc.

productVersion
The exact product version.

project
The project the artifact belong to.

resourceName
The name of the artifact.

resourceType
The type of the artifact.

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 version is not found.
```

500

```
An internal error occurred.
```

Application Version (DELETE)

Deletes a specific application 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 by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/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.

`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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

`locale`
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The metadata of the deleted application version.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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="8.0">
  <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
      name="myname"
      type="mytype"/>
    <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:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 version is not found.`

500

`An internal error occurred.`

Audit (GET)

Returns Audit Information.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/audit

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/audit?fromDate=2016-03-01&toDate=2016-03-10
```

Query Parameters

Query parameters are optional.

fromDate
Specify from which date audit log is required

toDate
Specify till which date audit log is required

Produces

application/zip

Errors

400
`Invalid payload.`

500
`An internal error occurred.`

Confidential Clients (GET)

Retrieves the confidential clients list of a specific runtime.

Roles

Users in the following roles are authorized to perform this operation:
- **`mfpadmin`**
- **`mfpdeployer`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/confidentialclients

Example

`https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/confidentialclients?locale=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 confidential clients list of the specified runtime.

JSON Example

```
{
  "clients" : [
    {
      "allowedScope" : "clients:read-public clients:read-protected update",
      "displayName" : "My Client",
      "id" : "ABC",
      "secret" : "12345",
    },
    ...
  ],
  "productVersion" : "8.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<confidential-clients productVersion="8.0">
  <clients>
    <client
      allowedScope="clients:read-public clients:read-protected update"
      displayName="My Client"
      id="ABC"
      secret="12345"/>
    ...
  </clients>
</confidential-clients>
```

Response Properties

The response has the following properties:

clients
The confidential clients of the runtime.

productVersion
The exact product version.

The *confidential client* has the following properties:

allowedScope
The allowed scope of the client.

displayName
The display name of the client.

id The identifier of the client.

secret
The secret of the confidential client.

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime is not found.`

500

`An internal error occurred.`

Confidential Clients (PUT)

Sets the confidential clients list of a specific runtime.

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:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/confidentialclients

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/confidentialclients?async=fa
```

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

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "clients" : [
    {
      "allowedScope" : "clients:read-public clients:read-protected update",
      "displayName" : "My Client",
      "id" : "ABC",
      "secret" : "12345",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<confidential-clients>
  <clients>
    <client
      allowedScope="clients:read-public clients:read-protected update"
      displayName="My Client"
      id="ABC"
      secret="12345"/>
    ...
  </clients>
</confidential-clients>
```

Payload Properties

The payload has the following properties:

clients
The confidential clients of the runtime.

The *confidential client* has the following properties:

allowedScope
The allowed scope of the client.

displayName
The display name of the client.

id The identifier of the client.

secret
The secret of the confidential client.

Response

The confidential clients of the specified runtime.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-confidential-clients-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-confidential-clients-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 confidential clients.

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_ARTIFACT`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 is not found.`

500

`An internal error occurred.`

Create Subscription (POST)

Creates a new subscription for a tag.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

POST

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

action
When set to `delete`, this parameter unsubscribes a device from the list of tags that is specified in the `tagNames` field of the JSON body.

locale
The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "deviceId" : "12345-6789",
  "tagName" : "SampleTag",
}
```

Payload Properties

The payload has the following properties:

deviceId
The unique identifier of the device

tagName
The tag name to subscribe.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Create Tag (POST)

Creates a tag with a unique name in the application that is referenced by the `applicationId` parameter.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

```
POST
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "description" : "This is a sample tag.",
  "name" : "SampleTag",
}
```

Payload Properties

The payload has the following properties:

description
The description of the tag.

name
The name of the tag.

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

500

```
An internal error occurred.
```

Delete APNs settings (DELETE)

Deletes the APNs settings to the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/apnsConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

Query Parameters

Query parameters are optional.

`locale`
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Delete GCM settings (DELETE)

Deletes the GCM settings to the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/gcmConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

500

```
An internal error occurred.
```

Delete WNS settings (DELETE)

Deletes the WNS settings from the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/wnsConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

Query Parameters

Query parameters are optional.

locale
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

500

```
An internal error occurred.
```

Delete Message (DELETE)

Deletes a message identified by the messageId parameter.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/messages/*message-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

message-id
: The message id of push message in push server

Query Parameters

Query parameters are optional.

locale
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Delete Subscription (DELETE)

Unsubscribes the device from the tag by using the subscription identifier. This method deletes neither the device registration nor the tag.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions/*subscription-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

`subscription-id`
The subscription id of the application register with Push

Produces

application/json, application/xml, text/xml

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

500

```
An internal error occurred.
```

Delete Tag (DELETE)

Deletes the tag in the application.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags/*tag-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

Produces

application/json, application/xml, text/xml

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Deploy (POST)

Deploys a multipart compressed file.

Description

A deployable can contains an adapter, application, license configuration, keystore, web resource, etc.

The method first checks whether the input deployable is valid. Then, the method transfers the deployable 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 by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

```
POST
```

Path

/management-apis/2.0/runtimes/*runtime-name*/deploy/multi

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/deploy/multi?async=false&l
```

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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

`locale`
The locale used for error messages.

Consumes

multipart/form-data

Produces

application/json, application/xml, text/xml

Response

The metadata of the deployable.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</deploy-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 deployable.

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_ARTIFACT`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 deployable 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.`

Deploy Application Authenticity Data (POST)

Deploys application authenticity data for a specific application version.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/ *application-env*/*application-version*/authenticity

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/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.

`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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

`locale`
The locale used for error messages.

Consumes

multipart/form-data

Produces

application/json, application/xml, text/xml

Response

The metadata of the deployable.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-authenticitydata-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-appversion-authenticitydata-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 deployable.

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_ARTIFACT`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 deployable data is provided.`

403
`The user is not authorized to call this service.`

404
`The corresponding runtime version is not found or not running.`

500
`An internal error occurred.`

Deploy a web resource (POST)

deploy a web resource zip for a specific application version.

Roles

Users in the following roles are authorized to perform this operation:
- **`mfpadmin`**
- **`mfpdeployer`**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*/web

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/android/1.0/web?async=false&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.

`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

multipart/form-data

Produces

application/json, application/xml, text/xml

Response

The metadata of the web resource .

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-webresources-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-appversion-webresources-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 web resource.

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_ARTIFACT`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 deployable data is provided.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime version is not found or not running.`

500

`An internal error occurred.`

Device Application Status (PUT)

Changes the status of a specific application on a specific device.

Description

An application 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 the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/devices/*device-id*/applications/*application-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/devices/12345-6789/applicati
```

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`. By default, transactions are processed in synchronous mode.

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" : "8.0",
  "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="8.0">
  <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 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 the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpoperator`**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/devices/*device-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/devices/12345-6789?async=fal
```

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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

`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" : "8.0",
  "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="8.0">
  <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.`

Device (DELETE)

Deletes all metadata of a specific device.

Description

This transaction can run synchronously or asynchronously. If the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*/devices/*device-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/devices/12345-6789?async=f
```

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. The allowed values are `true` and `false`. By default, transactions are processed in synchronous mode.

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" : "8.0",
  "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="8.0">
  <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.
```

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:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/devices

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/devices?bookmark=ABC&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.

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" : [
        {
          "appId" : "com.ibm.app$ios$1.0.0",
          "appName" : "myapplication",
          "certSerialNumber" : "",
          "deviceId" : "12345-6789",
          "deviceStatus" : "LOST",
          "status" : "ENABLED",
        },
        ...
      ],
      "deviceDisplayName" : "Jeremy's Personal Phone",
      "deviceModel" : "Nexus 7",
      "deviceOs" : "4.4",
      "id" : "12345-6789",
      "lastAccessed" : "2014-05-13T00:18:36.979Z",
      "status" : "LOST",
      "userIds" : [
        {
          "str" : "Jeremy",
        },
        ...
      ],
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "8.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<devices
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="8.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deviceDisplayName="Jeremy's Personal Phone"
      deviceModel="Nexus 7"
      deviceOs="4.4"
      id="12345-6789"
      lastAccessed="2014-05-13T00:18:36.979Z"
      status="LOST">
      <applicationDeviceAssociations>
        <applicationDeviceAssociation
          appId="com.ibm.app$ios$1.0.0"
          appName="myapplication"
          certSerialNumber=""
          deviceId="12345-6789"
          deviceStatus="LOST"
          status="ENABLED"/>
        ...
      </applicationDeviceAssociations>
      <userIds>
        <userId str="Jeremy"/>
        ...
      </userIds>
    </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 one page of devices is returned.

pageNumber
The page index if only one page of devices is returned.

pageSize
The page size if only one page of devices is returned.

prevPageBookmark
The bookmark of the previous or first page if only one page of devices is returned.

productVersion
The exact product version.

startIndex
The start index in the total list if only one page of devices is returned.

totalListSize
The total number of devices.

The *device* has the following properties:

applicationDeviceAssociations
The applications on the device.

deviceDisplayName
The friendly name of the device.

deviceModel
The device model.

deviceOs
The device operating system.

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`.

userIds
The applications on the device.

The *device application* has the following properties:

appId
The application id.

appName
The name of the application.

certSerialNumber
The serial number of the certificate

deviceId
The device id.

deviceStatus
The status of the device:`ACTIVE`, `LOST`, `STOLEN`, `EXPIRED`, `DISABLED`.

status
The status of the application: `ENABLED` or `DISABLED`.

The *user-ids* has the following properties:

str
The name of the user

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.`

Diagnostic Service (GET)

Retrieves diagnostic information for administration, runtime, configuration (live update), and push services.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/diagnostic

Example

`https://www.example.com/mfpadmin/management-apis/2.0/diagnostic`

Produces

application/json, application/xml, text/xml

Response

Information about the diagnostic.

JSON Example

```
{
  "adminDB" : {
    "status" : "available",
  },
  "analyticsService" : [
    {
      "runtime" : "mfp",
      "status" : "available",
    },
    ...
  ],
  "configService" : {
    "status" : "available",
  },
  "productVersion" : "8.0",
  "pushDiagnostic" : {
    "status" : "available",
  },
  "runningProjectStatuses" : {
    "hasAppsOrAdapter" : true,
    "name" : "mfp",
    "running" : true,
    "synchronized" : "ok",
  },
  "runtimeDiagnostic" : [
    {
      "instances" : [
        {
          "Providers" : [
            {
              "DatabaseDiagnosticBean" : {
                "message" : "Database is OK",
                "ok" : "Ok",
              },
            },
            ...
          ],
          "overallStatus" : "Ok",
        },
        ...
      ],
      "name" : "mfp",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<diagnostic-info productVersion="8.0">
  <adminDB status="available"/>
  <analyticsServiceArray>
    <analyticsService
      runtime="mfp"
      status="available"/>
    ...
  </analyticsServiceArray>
  <configService status="available"/>
  <pushDiagnostic status="available"/>
  <runningProjectStatuses
    hasAppsOrAdapter="true"
    name="mfp"
    running="true"
    synchronized="ok"/>
  <runtimeDiagnosticArray>
    <runtimeDiagnostic name="mfp">
      <instances>
        <instance overallStatus="Ok">
          <Providers>
            <Provider>
              <DatabaseDiagnosticBean
                message="Database is OK"
                ok="Ok"/>
            </Provider>
            ...
          </Providers>
        </instance>
        ...
      </instances>
    </runtimeDiagnostic>
    ...
  </runtimeDiagnosticArray>
</diagnostic-info>
```

Response Properties

The response has the following properties:

adminDB
  Status of the administration database.

analyticsService
  Status of the Analytics service.

configService
  Status of the configuration service/live update service.

productVersion
  The exact product version.

pushDiagnostic
  Status of the push service.

runningProjectStatuses
  Status of all the running projects.

runtimeDiagnostic
  Runtime diagnostics of each runtime.

The *admin-db* has the following properties:

status
  Status of the administration database.

The *analytics-service* has the following properties:

runtime
  The name of the runtime.

status
  Status of the Analytics service.

The *config-service* has the following properties:

status
  Status of the configuration service/live update service.

The *push-service* has the following properties:

status
  Status of the push service.

The *running-project* has the following properties:

hasAppsOrAdapter
  Whether the project includes any applications or adapters

name
  The name of the project

running
  Whether the project is running

synchronized
Synchronization status of the project

The *runtime-diagnostic* has the following properties:

instances
Status of each runtime instance

name
The name of the runtime

The *runtime-status* has the following properties:

Providers
Status of each runtime instance

overallStatus
The overall status of the runtime

The *providers-status* has the following properties:

DatabaseDiagnosticBean
Database Diagnostics Bean status

The *db-status* has the following properties:

message
Database status message

ok Database status

Errors

403

`The user is not authorized to call this service.`

500

`An internal error occurred.`

Export adapter resources (GET)

Retrieves a compressed file that contains all or selected resources for specific adapters for this runtime.

Description

The method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/export/adapters/*adapter-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/adapters/myadapter?inc
```

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.

`include`
Optional query parameter to get selected resources.

`locale`
The locale used for error messages.

Produces

application/octet-stream

Response

The compressed file containing all or selected resources for a specific adapter for this runtime.

Errors

400

`The request is invalid.`

403

`The user is not authorized to call this service.`

404

`One or more of the corresponding binary files were not found.`

416

`The requested range of bytes is not valid.`

500

`An internal error occurred.`

Export adapters (GET)

Retrieves a compressed file that contains all or selected adapter resources for this runtime.

Description

The method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/export/adapters

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/adapters?include=AD
```

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.

`include`
: Optional query parameters to get selected resources.

`locale`
: The locale used for error messages.

Produces

application/octet-stream

Response

The compressed file containing all or selected adapter resources for this runtime.

Errors

400

`The request is invalid.`

403

`The user is not authorized to call this service.`

404

`One or more of the corresponding binary files were not found.`

416

`The requested range of bytes is not valid.`

500

`An internal error occurred.`

Export application environment (GET)

Retrieves a compressed binary file that contains all or selected application environment-specific resources for this runtime.

Description

The method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/export/applications/*application-name*/*application-env*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/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.

Query Parameters

Query parameters are optional.

`include`
: Optional query parameter to get selected resources.

`locale`
: The locale used for error messages.

Produces

application/octet-stream

Response

The compressed file containing all or selected application environment-specific resources for this runtime.

Errors

400

```
The request is invalid.
```

403

```
The user is not authorized to call this service.
```

404

```
One or more of the corresponding binary files were not found.
```

416

```
The requested range of bytes is not valid.
```

500

```
An internal error occurred.
```

Export application environment resources (GET)

Retrieves a compressed binary resource for a specific version of an application environment for this runtime.

Description

The method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/export/applications/*application-name*/*application-env*/*application-version*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/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.

include
Optional query parameter to get selected resources.

locale
The locale used for error messages.

Produces

application/octet-stream

Response

The compressed binary data, containing all or selected resources for a specific version of an application environment for this runtime.

Errors

400

```
The request is invalid.
```

403

```
The user is not authorized to call this service.
```

404

```
One or more of the corresponding binary files were not found.
```

416

```
The requested range of bytes is not valid.
```

500

```
An internal error occurred.
```

Export application resources (GET)

Retrieves a compressed file that contains all or selected application-specific resources for this runtime.

Description

The method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/export/applications/*application-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/applications/myappl
```

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.

`include`
Optional query parameter to get selected resources.

`locale`
The locale used for error messages.

Produces

application/octet-stream

Response

The compressed file containing all or selected application-specific resources for this runtime.

Errors

400

`The request is invalid.`

403

`The user is not authorized to call this service.`

404

`One or more of the corresponding binary files were not found.`

416

`The requested range of bytes is not valid.`

500

`An internal error occurred.`

Export applications (GET)

Retrieves a compressed file that contains all or selected application resources for this runtime.

Description

The method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/export/applications

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/applications?include=
```

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.

include
Optional query parameter to get selected resources.

locale
The locale used for error messages.

Produces

application/octet-stream

Response

The compressed file containing all or selected application resources for this runtime.

Errors

400

`The request is invalid.`

403

`The user is not authorized to call this service.`

404

`One or more of the corresponding binary files were not found.`

416

`The requested range of bytes is not valid.`

500

`An internal error occurred.`

Export resources (GET)

Retrieves a compressed file (.zip) that contains all the specified resources.

Description

It supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

- **mfpmonitor**
- **mfpoperator**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*/export

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export?locale=de_DE&resourceI
```

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.

resourceInfos
The information to uniquely identify a resource. Format resourceName||resourceType. For Adapter: {adapterName}||ADAPTER_CONTENT For Application Descriptor: {appName${platform}${version}||APP_DESCRIPTOR For Licence Configuration: {appName}||APP_LICENSE_CONFIG For Application Configuration: {appName${platform}${version}||APP_USER_CONFIGURATION For Keystore: keystore||KEYSTORE For Web Resource: {appName${platform}${version}||APP_WEB_CONTENT

Produces

application/octet-stream

Response

The compressed file of the specified deployables.

Errors

400

```
The request is invalid.
```

403

```
The user is not authorized to call this service.
```

404

```
One or more of the corresponding binary files were not found.
```

416

`The requested range of bytes is not satisfiable.`

500

`An internal error occurred.`

Export runtime resources (GET)

Retrieves a compressed file that contains all or selected runtime-specific resources.

Description

This method supports range requests to deliver only a range of the bytes of the binary file. Clients can use this feature to resume a download after an interruption.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/export/all

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/export/all?include=RUNTIME
```

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.

`include`
Optional query parameter to get selected resources.

`locale`
The locale used for error messages.

Produces

application/octet-stream

Response

The compressed file containing all or selected runtime-specific resources.

Errors

400
`The request is invalid.`

403
`The user is not authorized to call this service.`

404
`One or more of the corresponding binary files were not found.`

416
`The requested range of bytes is not valid.`

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:
- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/farm

Example

`https://www.example.com/mfpadmin/management-apis/2.0/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 removes a farm node only if it is marked as Down. If you want to force the deletion, even if the farm member is marked as Alive, set the `force` argument to `true`.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*/farm/*server-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/farm/farm_member_1?force=f
```

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 value 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.
```

Get Message (GET)

Retrieves information about a message identified by its messageId parameter.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/messages/*message-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

`message-id`
The message id of push message in push server

Query Parameters

Query parameters are optional.

`locale`
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

Retrieves the meta data of the message identified by the messageId parameter.

JSON Example

```
{
  "alert" : "New update available",
  "messageId" : "New update available",
  "productVersion" : "8.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-messages
  alert="New update available"
  messageId="New update available"
  productVersion="8.0"/>
```

Response Properties

The response has the following properties:

`alert`
: A string to be displayed in the alert.

`messageId`
: The identifier of the notification message sent.

`productVersion`
: The exact product version.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Get Tags (GET)

Retrieves all or a subset of tags in the application.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

expand
Retrieves additional metadata for every subscription that is returned in the response.

filter
The filter specifies the search criteria. Refer to the filter section for the detailed syntax.

locale
The locale used for error messages.

offset
The pagination offset that is normally used in association with the size.

size
The pagination size that is normally used in association with the offset to retrieve a subset.

subscriptionCount
If this parameter is set to `true`, the method retrieves the number of subscriptions for each platform.

Produces

application/json, application/xml, text/xml

Response

Retrieves tags of the application.

JSON Example

```
{
  "productVersion" : "8.0",
  "tags" : {
    "createdMode" : "API",
    "createdTime" : "2016-03-19T06:34:42Z",
    "description" : "This is a sample tag",
    "href" : "http://localhost:9080/imfpush/v1/apps/com.test.one/tags/SampleTag",
    "lastUpdatedTime" : "2016-03-22T06:34:42Z",
    "name" : "SampleTag",
    "uri" : "http://localhost:9080/imfpush/v1/apps/com.test.one/tags/SampleTag",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-tags productVersion="8.0">
  <tags
    createdMode="API"
    createdTime="2016-03-19T06:34:42Z"
    description="This is a sample tag"
    href="http://localhost:9080/imfpush/v1/apps/com.test.one/tags/SampleTag"
    lastUpdatedTime="2016-03-22T06:34:42Z"
    name="SampleTag"
    uri="http://localhost:9080/imfpush/v1/apps/com.test.one/tags/SampleTag"/>
</push-tags>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

tags
The list of tags of the application.

The *push tags* has the following properties:

createdMode
How the tag was created. The possible values are `UI` or `API`.

createdTime
The time at which the tag was created.

description
The description of the tag.

href
The link to the tag.

lastUpdatedTime
The time at which the tag was last updated.

name
The name of the tag.

uri
The link to the tag.

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

500

```
An internal error occurred.
```

Get APNs Settings (GET)

Retrieves APNs credentials for the application.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/apnsConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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 APNS certificate.

JSON Example

```
{
  "certificate" : "apns-certificate-sandbox.p12",
  "isSandBox" : true,
  "productVersion" : "8.0",
  "validUntil" : "2016-09-10T09:32:30.000Z",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushAPNS
  certificate="apns-certificate-sandbox.p12"
  isSandBox="true"
  productVersion="8.0"
  validUntil="2016-09-10T09:32:30.000Z"/>
```

Response Properties

The response has the following properties:

`certificate`
: The name of the APNS certificate

`isSandBox`
: Is this certificate for SandBox or Production?

`productVersion`
: The exact product version.

`validUntil`
: The expiration date for the certificate

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Get GCM Settings (GET)

Retrieves GCM credentials for the application.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/gcmConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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 GCM credentials.

JSON Example

```
{
  "apiKey" : "AIzaSyBnWWReKAFrOPiw75QQAcRM",
  "productVersion" : "8.0",
  "senderId" : "11639055112",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushGCM
  apiKey="AIzaSyBnWWReKAFrOPiw75QQAcRM"
  productVersion="8.0"
  senderId="11639055112"/>
```

Response Properties

The response has the following properties:

`apiKey`
: GCM Api Key

`productVersion`
: The exact product version.

`senderId`
: The project ID that is signed up at the Google API console.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Get WNS Settings (GET)

Retrieves WNS credentials for the application.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/wnsConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

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 WNS certificate.

JSON Example

```
{
  "clientSecret" : "712345dummyvalues12345",
  "packageSID" : "ms-app://s-1-15-2-dummyvalues12345",
  "productVersion" : "8.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushWNS
  clientSecret="712345dummyvalues12345"
  packageSID="ms-app://s-1-15-2-dummyvalues12345"
  productVersion="8.0"/>
```

Response Properties

The response has the following properties:

clientSecret
The Secret Key

packageSID
Package Security Identifier (SID)

productVersion
The exact product version.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Global Configuration (GET)

Retrieves information about the global configuration.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/config?locale=de_DE
```

Query Parameters

Query parameters are optional.

`locale`
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The global configuration.

JSON Example

```
{
  "analyticsConsoleUrl" : [
    {
      "runtime" : "myruntime",
      "url" : "https://www.example.com/analytics/console",
    },
    ...
  ],
  "auditEnabled" : true,
  "cloudantDashboardUrl" : "https://example.cloudant.com/dashboard.html",
  "iosEdition" : false,
  "productVersion" : "8.0",
  "pushConfidentialClientsStatus" : "ok",
  "pushEnabled" : true,
  "swaggerUrl" : "https://www.example.com/doc/?url=https://www.example.com/api/adapterdoc/sampleAd
  "topology" : "STANDALONE",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<global-config
  auditEnabled="true"
  cloudantDashboardUrl="https://example.cloudant.com/dashboard.html"
  iosEdition="false"
  productVersion="8.0"
  pushConfidentialClientsStatus="ok"
  pushEnabled="true"
  swaggerUrl="https://www.example.com/doc/?url=https://www.example.com/api/adapterdoc/sampleAdapter"
  topology="STANDALONE">
  <analyticsConsoleUrls>
    <analyticsConsoleUrl
      runtime="myruntime"
      url="https://www.example.com/analytics/console"/>
    ...
  </analyticsConsoleUrls>
</global-config>
```

Response Properties

The response has the following properties:

analyticsConsoleUrl
The array of Analytics console URLs for available runtimes

auditEnabled
Whether audit is enabled.

cloudantDashboardUrl
The link to the Cloudant dashboard, if any.

iosEdition
Whether the server is an iOS Edition.

productVersion
The exact product version.

pushConfidentialClientsStatus
Status of the internal push and administrative confidential client

pushEnabled
Whether the push service is enabled.

swaggerUrl
The link to the Swagger UI URL.

topology
Server topology. Possible values: "STANDALONE", "CLUSTER" or "FARM"

The *analytics-urls* has the following properties:

runtime
The name of the runtime.

url
The URL of the Analytics console.

Errors

403

`The user is not authorized to call this service.`

500

```
An internal error occurred.
```

Keystore (GET)

Retrieves keystore properties for a deployed keystore of a runtime.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/keystore

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/keystore
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

Produces

application/json, application/xml, text/xml

Response

The keystore properties as JSON code.

JSON Example

```
{
  "keystore.password" : "password",
  "keystore.type" : jks,
  "productVersion" : "8.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<runtime
  keystore.password="password"
  keystore.type="jks"
  productVersion="8.0"/>
```

Response Properties

The response has the following properties:

`key.alias`
The alias of the entry where the private key and certificate are stored, in the keystore.

`key.alias.password`
The password to the alias in the keystore.

`keystore.password`
The password to the keystore.

`keystore.type`
The type of the keystore. Valid values are jks or pkcs12..

`productVersion`
The exact product version.

Errors

403

`The user is not authorized to call this service.`

404

`The resource is not found.`

500

`An internal error occurred.`

Keystore (POST)

Deploy a keystore for a runtime.

Description

A deployable is a keystore.

The method first checks whether the input deployable is valid. Then, the method transfers the deployable to the database and to the runtime.

This transaction can run synchronously or asynchronously. If the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/keystore

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/keystore?async=false&local
```

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: `true` and `false`. By default, transactions are processed in synchronous mode.

locale
: The locale used for error messages.

type
: The type of the deployable is RUNTIME_KEYSTORE.

Consumes

multipart/form-data

Produces

application/json, application/xml, text/xml

Response

The metadata of the deployable.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_ARTIFACT",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ARTIFACT"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</deploy-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 deployable.

`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_ARTIFACT`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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

`Invalid payload.`

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.`

Keystore (DELETE)

Deletes a keystore from the runtime.

Description

This transaction can run synchronously or asynchronously. If the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*/keystore

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/keystore?async=false&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.

async
Whether the transaction is processed synchronously or asynchronously. Allowed values: `true` and `false`. By default, transactions are processed in synchronous mode.

locale
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The metadata of the deleted keystore.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_KEYSTORE",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-keystore-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="DELETE_KEYSTORE"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</delete-keystore-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 keystore.

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_KEYSTORE.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 keystore for the runtime mfp does not exist in the MobileFirst administration database. The data
```

500

```
An internal error occurred.
```

License configuration (DELETE)

Deletes a license configuration for the application name.

Description

This transaction can run synchronously or asynchronously. If the transaction is processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later by using the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/license/*application-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/license/myapplication?asyn
```

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: `true` and `false`. By default, transactions are processed in synchronous mode.

locale
: The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The metadata of the deleted license configuration.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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_LICENSE_CONFIG",
    "userName" : "demouser",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-app-licenseconfig-result
  ok="false"
  productVersion="8.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="DELETE_LICENSE_CONFIG"
    userName="demouser">
    <description
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</delete-app-licenseconfig-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 license configuration.

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_LICENSE_CONFIG`.

userName
The user that initiated the transaction.

The *description* has the following properties:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 version is not found.`

500

`An internal error occurred.`

Push Device Registration (GET)

Retrieves all or a subset of existing device registrations to the push service.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**

- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/devices

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

expand
Whether to retrieve detailed information about the device.

filter
The search criteria.

locale
The locale used for error messages.

offset
From where to start listing entries, depending on the value of the `size` parameter.

size
The maximum number of entries to be listed per page. For example: 10.

userId
The user identifier of the device.

Produces

application/json, application/xml, text/xml

Response

Retrieves all or a subset of existing device registration to the push service.

JSON Example

```
{
  "devices" : [
    {
      "deviceId" : "JeremyiOSPhone",
      "href" : "http://localhost:9080/imfpush/v1/apps/com.test.one/devices/JeremyiOSPhone",
      "userId" : "Jeremy",
    },
    ...
  ],
  "productVersion" : "8.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-devices productVersion="8.0">
  <devices>
    <device
      deviceId="JeremyiOSPhone"
      href="http://localhost:9080/imfpush/v1/apps/com.test.one/devices/JeremyiOSPhone"
      userId="Jeremy"/>
    ...
  </devices>
</push-devices>
```

Response Properties

The response has the following properties:

devices
    The list of devices registered with the application

productVersion
    The exact product version.

The *device-list* has the following properties:

deviceId
    The unique identifier of the device.

href
    The link to the device identifier

userId
    The user identifier of the device.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

```
An internal error occurred.
```

Push Device Registration (DELETE)

Deletes(unregisters) an existing device registration from the push service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/devices/*device-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

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.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

Query Parameters

Query parameters are optional.

`deviceId`
: Retrieves subscriptions only for the specified device.

`expand`
: Retrieves additional metadata for every subscription that is returned in the response.

`filter`
: The filter specifies the search criteria. Refer to the filter section for the detailed syntax.

`locale`
: The locale used for error messages.

offset
The pagination offset that is normally used in association with the page size.

size
The pagination size that is normally used in association with the offset to retrieve a subset.

tagName
Retrieves subscriptions only for the specified tag.

userId
Retrives subscriptions only for the specified user.

Produces

application/json, application/xml, text/xml

Response

Retrieves all push subscriptions for the application.

JSON Example

```
{
  "productVersion" : "8.0",
  "subscriptions" : {
    "deviceId" : "12345-6789",
    "href" : "http://localhost:9080/imfpush/v1/apps/com.test.one/subscriptions/2",
    "subscriptionId" : "12",
    "tagName" : "SampleTag",
    "userId" : "Jeremy",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-subsciptions productVersion="8.0">
  <subscriptions
    deviceId="12345-6789"
    href="http://localhost:9080/imfpush/v1/apps/com.test.one/subscriptions/2"
    subscriptionId="12"
    tagName="SampleTag"
    userId="Jeremy"/>
</push-subsciptions>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

subscriptions
The list of push subscriptions.

The *push subcriptions* has the following properties:

deviceId
The unique identifier of the device.

href
The link to the subscription.

subscriptionId
The unique identifier of the subscription.

tagName
The tag name for which to retrieve subscriptions.

userId
The user identifier for which to retrieve subscriptions.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Register Application with Push Service (POST)

Registers an application with the push server.

Description

Creates a new server application for a push service. The applicationId is a unique identifier for this application. The application is a parent resource for devices, subscriptions, tags, and messages. The application must be created before it can access any of the child resources. If the application is deleted, all the children are deleted. The application holds the configurations, such as the Apple Push Notification Service (APNS) or Google Cloud Message (GCM) configuration, which is required by the push service to send messages. The API first creates the application and then sets the APNS and GCM credentials.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications
```

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

JSON Example

```
{
  "Enabled" : true,
  "applicationId" : "com.sample.bankApp",
}
```

Payload Properties

The payload has the following properties:

Enabled
Whether the application is enabled or disabled.

applicationId
The bundleId/PackageName/ProjectIdentityName of the application

Errors

400

```
The request was not understood by the push server. An invalid JSON object could result in this error
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

409

```
An application with the specified identifier already exists.
```

415

```
Unsupported Media Type - The content type specified in Content-Type header is not application/json.
```

500

```
An internal error occurred.
```

Remove Subscription (DELETE)

Unsubscribes the specified device from a tag.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**

Method

```
DELETE
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

Query Parameters

Query parameters are optional.

deviceId
The unique ID for the device

locale
The locale used for error messages.

tagName
The name of the tag to unsubscribe from

Produces

application/json, application/xml, text/xml

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

`The corresponding runtime or application is not found or not running.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Retrieve Device Registration (GET)

Retrieves an existing device registration to the push service.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/devices/*device-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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

Retrieves an existing device registration of push.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2016-03-15T16:30:19Z",
  "deviceId" : "JeremyiOSPhone",
  "href" : "http://localhost:9080/imfpush/v1/apps/com.test.one/devices/JeremyiOSPhone",
  "lastUpdatedTime" : "2016-03-18T16:30:19Z",
  "platform" : "A",
  "productVersion" : "8.0",
  "token" : "c6a41224 23333917 9fde1532",
  "userId" : "Jeremy",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-device
  createdMode="API"
  createdTime="2016-03-15T16:30:19Z"
  deviceId="JeremyiOSPhone"
  href="http://localhost:9080/imfpush/v1/apps/com.test.one/devices/JeremyiOSPhone"
  lastUpdatedTime="2016-03-18T16:30:19Z"
  platform="A"
  productVersion="8.0"
  token="c6a41224 23333917 9fde1532"
  userId="Jeremy"/>
```

Response Properties

The response has the following properties:

`createdMode`
The mode of device creation

`createdTime`
The time at which the device registration occurred

`deviceId`
The unique identifier of the device

`href`
The link to the device identifier

`lastUpdatedTime`
Last update time

`platform`
The device platform

`productVersion`
The exact product version.

`token`
The unique push token of the device

`userId`
The user identifier of the device.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Retrieve Tag (GET)

Retrieves the specified tag in the application.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags/*tag-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

Produces

application/json, application/xml, text/xml

Response

Retrieves details of a specific tag of the application.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2016-03-19T06:34:42Z",
  "description" : "This is a Sample tag",
  "lastUpdatedTime" : "2016-03-22T06:34:42Z",
  "name" : "SampleTag",
  "productVersion" : "8.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-tag
  createdMode="API"
  createdTime="2016-03-19T06:34:42Z"
  description="This is a Sample tag"
  lastUpdatedTime="2016-03-22T06:34:42Z"
  name="SampleTag"
  productVersion="8.0"/>
```

Response Properties

The response has the following properties:

createdMode
How the tag was created. The possible values are `UI` or `API`.

createdTime
The time at which the tag was created.

description
The description of the tag.

lastUpdatedTime
The time at which the tag was last updated.

name
The name of the tag.

productVersion
The exact product version.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Retrieve Web Resource (GET)

Retrieves the metadata of a web resource for a specific application version.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*/applications/*application-name*/*application-env*/*application-version*/web

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/applications/myapplication/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.

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 metadata of the web resource for the specified application version.

JSON Example

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "displayName" : "MyApplication",
  "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applicati
  "productVersion" : "8.0",
  "project" : {
    "name" : "myproject",
  },
  "resourceName" : "abc",
  "resourceType" : "APP_DESCRIPTOR",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<webresource
  deployTime="2014-04-13T00:18:36.979Z"
  displayName="MyApplication"
  link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/{runtime-name}/applications/
  productVersion="8.0"
  resourceName="abc"
  resourceType="APP_DESCRIPTOR">
  <project name="myproject"/>
</webresource>
```

Response Properties

The response has the following properties:

deployTime
The date in ISO 8601 format when the artifact was deployed.

displayName
The optional display name of the artifact.

link
The URL to access detailed information about the deployed artifacts such as application, adapter etc.

productVersion
The exact product version.

project
The project the artifact belong to.

resourceName
The name of the artifact.

resourceType
The type of the artifact.

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.
```

Retrieve Subscription to Push Service. (GET)

The subscription referenced by the subscription identifier is retrieved.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions/*subscription-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

subscription-id
: The subscription id of the application register with Push

Produces

application/json, application/xml, text/xml

Response

Retrieves all push subscriptions for the application.

JSON Example

```
{
  "deviceId" : "12345-6789",
  "href" : "http://localhost:9080/imfpush/v1/apps/com.test.one/subscriptions/2",
  "productVersion" : "8.0",
  "subscriptionId" : "12",
  "tagName" : "SampleTag",
  "userId" : "Jeremy",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<push-subsciption
  deviceId="12345-6789"
  href="http://localhost:9080/imfpush/v1/apps/com.test.one/subscriptions/2"
  productVersion="8.0"
  subscriptionId="12"
  tagName="SampleTag"
  userId="Jeremy"/>
```

Response Properties

The response has the following properties:

deviceId
: The unique identifier of the device.

href
: The link to the subscription.

productVersion
: The exact product version.

subscriptionId
: The unique identifier of the subscription.

tagName
: The tag name for which to retrieve subscriptions.

userId
: The user identifier for which to retrieve subscriptions.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Runtime Configuration (GET)

Retrieves the user configuration of a specific runtime.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/config?flattened=false&local
```

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.

flattened
: If true (default), the configuration is a flat list of properties, otherwise a hierarchy of objects.

locale
: The locale used for error messages.

mode
: If no mode is specified, it returns the current user configuration. If the mode `defaults` is specified, it returns the default configuration.

Produces

application/json, application/xml, text/xml

Response

The user configuration of the specified runtime.

JSON Example

```
{
  "adapters" : {
    "compressResponseThreshold" : {
      "value" : 20480,
    },
  },
  "analytics" : {
    "additionalPackages" : {
      "value" : "com.admin.util",
    },
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<runtimeconfig>
  <adapters>
    <compressResponseThreshold value="20480"/>
  </adapters>
  <analytics>
    <additionalPackages value="com.admin.util"/>
  </analytics>
</runtimeconfig>
```

Response Properties

The response has the following properties:

adapters
: The runtime properties for adapter.

analytics
: The runtime properties for analtyics

The *adapter-property* has the following properties:

compressResponseThreshold
: Compression threshold, in bytes, from which the server tries to compress the MobileFirst adapter response if the client accepts gzip.

The *compressthreshold* has the following properties:

value
: The value of the compression threshold

The *analytics-property* has the following properties:

additionalPackages
: A comma-separated list of packages that the logger uses to generate the output sent to the MobileFirst Analytics server.

The *additional package* has the following properties:

value
: The list of packages that the logger uses

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime is not found.`

500

`An internal error occurred.`

Runtime configuration (PUT)

Sets the user configuration of a specific runtime.

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:

- **mfpadmin**

- **mfpdeployer**

Method

PUT

Path

/management-apis/2.0/runtimes/*runtime-name*/config

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/config?async=false&locale=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.

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

```
{
  "adapters" : {
    "compressResponseThreshold" : {
      "value" : 20480,
    },
  },
  "analytics" : {
    "additionalPackages" : {
      "value" : "com.admin.util",
    },
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<runtimeconfig>
  <adapters>
    <compressResponseThreshold value="20480"/>
  </adapters>
  <analytics>
    <additionalPackages value="com.admin.util"/>
  </analytics>
</runtimeconfig>
```

Payload Properties

The payload has the following properties:

adapters
: The runtime properties for adapter.

analytics
: The runtime properties for analtyics

The *adapter-property* has the following properties:

compressResponseThreshold
: Compression threshold, in bytes, from which the server tries to compress the MobileFirst adapter response if the client accepts gzip.

The *compressthreshold* has the following properties:

value
: The value of the compression threshold

The *analytics-property* has the following properties:

additionalPackages
: A comma-separated list of packages that the logger uses to generate the output sent to the MobileFirst Analytics server.

The *additional package* has the following properties:

value
: The list of packages that the logger uses

Response

The configuration of the specified runtime.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "8.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "name" : "myname",
      "type" : "mytype",
    },
    "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-runtimeconfig-result
  ok="false"
  productVersion="8.0">
  <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
      name="myname"
      type="mytype"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-runtimeconfig-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:

contentNames
The optional names of the contained artifacts if multiple artifacts were deployed at once.

filename
The optional file name of the artifact.

name
The name of the artifact.

type
The type of the artifact.

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 is not found.`

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:

- **mfpadmin**

- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime?expand=true&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.

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

```
{
  "confidentialClients" : [
    {
      "allowedScope" : "**",
      "displayName" : "Test Client",
      "id" : "test",
      "secret" : "test",
    },
    ...
  ],
  "config" : {
  },
  "name" : "myruntime",
  "numberOfActiveDevices" : 100,
  "numberOfDecommisionedDevices" : 5,
  "productVersion" : "8.0",
  "running" : true,
  "runtimeInfo" : {
    "adaptersSecurityChecks" : {
    },
    "analytics" : {
      "analyticsEnabled" : {
        "defaultValue" : "true",
        "type" : boolean,
      },
    },
    "appAuthenticityEnabled" : true,
    "security" : {
      "activityUpdateThresholdSec" : {
        "defaultValue" : "3600",
        "type" : "integer",
      },
      "expirationMarginSec" : {
        "defaultValue" : "2",
        "type" : "integer",
      },
      "externalAZIntrospectionURL" : {
        "defaultValue" : "",
        "type" : "string",
      },
      "externalAZSharedSecret" : {
        "defaultValue" : "secret",
        "type" : "string",
      },
    },
    "securityChecks" : {
    },
  },
  "synchronizationStatus" : "ok",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<runtime
  name="myruntime"
  numberOfActiveDevices="100"
  numberOfDecommisionedDevices="5"
  productVersion="8.0"
  running="true"
  synchronizationStatus="ok">
  <confidentialClients>
    <confidentialClient
      allowedScope="**"
      displayName="Test Client"
      id="test"
      secret="test"/>
    ...
  </confidentialClients>
  <config/>
  <runtimeInfo appAuthenticityEnabled="true">
    <adaptersSecurityChecks/>
    <analytics>
      <analyticsEnabled
        defaultValue="true"
        type="boolean"/>
    </analytics>
    <security>
      <activityUpdateThresholdSec
        defaultValue="3600"
        type="integer"/>
      <expirationMarginSec
        defaultValue="2"
        type="integer"/>
      <externalAZIntrospectionURL
        defaultValue=""
        type="string"/>
      <externalAZSharedSecret
        defaultValue="secret"
        type="string"/>
    </security>
    <securityChecks/>
  </runtimeInfo>
</runtime>
```

Response Properties

The response has the following properties:

confidentialClients
The array of confidential clients registered with the runtime.

config
The runtime configurations

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.

productVersion
The exact product version.

running
Whether the runtime is currently active or has stopped.

runtimeInfo
The runtime Information

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.

The *conf-clients* has the following properties:

allowedScope
The allowed scopes

displayName
The display Name of the confidential client.

id The confidential client id.

secret
The secret of the confidential client.

The *runtime-info* has the following properties:

`adaptersSecurityChecks`
: The adapter security check information

`analytics`
: The analytics information

`appAuthenticityEnabled`
: Whether application authenticity is enabled.

`security`
: The security check information

`securityChecks`
: The security check information

The *analytics-check* has the following properties:

`analyticsEnabled`
: Analytics enabled

The *analytics-enabled* has the following properties:

`defaultValue`
: Analytics enabled

`type`
: The type

The *security-check* has the following properties:

`activityUpdateThresholdSec`
: Activity update threshold value

`expirationMarginSec`
: The expiration values in seconds

`externalAZIntrospectionURL`
: External AZ introspection URL

`externalAZSharedSecret`
: AZ shared secret

The *az-value1* has the following properties:

`defaultValue`
: The default value.

`type`
: The type

The *az-value2* has the following properties:

`defaultValue`
: The default value for shared secret

`type`
: The type

The *az-value3* has the following properties:

`defaultValue`
: The default value for activity update threshold

`type`
The type

The *az-value4* has the following properties:

`defaultValue`
The default value for external AZ introspection url

`type`
The type

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime is not found.`

500

`An internal error occurred.`

Runtime (DELETE)

Deletes a specific runtime.

Description

The purpose of this API is to allow to cleanup the 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:

- **`mfpadmin`**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime?locale=de_DE&mode=empty
```

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 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:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

`GET`

Path

/management-apis/2.0/runtimes/*runtime-name*/lock

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/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.
```

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:

- **`mfpadmin`**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*/lock

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/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.`

Runtimes (GET)

Retrieves metadata for the list of runtimes.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

`GET`

Path

/management-apis/2.0/runtimes

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes?locale=de_DE&mode=db
```

Query Parameters

Query parameters are optional.

`locale`
The locale used for error messages.

`mode`
The default mode `running` retrieves only the running runtimes, while the mode `db` retrieves also the runtimes stored in the database that might not be running.

Produces

application/json, application/xml, text/xml

Response

The metadata for the list of runtimes.

JSON Example

```
{
  "productVersion" : "8.0",
  "projects" : [
    {
      "apiVersion" : "2.0",
      "link" : "https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime",
      "name" : "myruntime",
      "running" : true,
      "synchronizationStatus" : "ok",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<projectconfiguration productVersion="8.0">
  <projects>
    <project
      apiVersion="2.0"
      link="https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime"
      name="myruntime"
      running="true"
      synchronizationStatus="ok"/>
    ...
  </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:

apiVersion
The API version

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.

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.

Errors

403

`The user is not authorized to call this service.`

500

`An internal error occurred.`

Send Bulk Messages (POST)

Send bulk messages by specifying various options.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

POST

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/messages/bulk

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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

JSON Example

```
{
  "ArrayOfMessageBody" : [
    {
      "message" : {
        "alert" : "New update available",
      },
      "settings" : {
        "apns" : {
          "delayWhileIdle" : true,
          "payload" : "",
          "sound" : "song.mp3",
          "timeToLive" : 100,
        },
        "gcm" : {
          "badge" : 1,
          "iosActionKey" : "Ok",
          "payload" : "",
          "sound" : "song.mp3",
          "type" : "SILENT",
        },
        "wns" : {
          "badge" : ,
          "cachePolicy" : false,
          "expirationTime" : 20,
          "raw" : ,
          "tile" : ,
          "toast" : ,
        },
      },
      "target" : {
        "deviceIds" : "[TestDeviceId,..]",
        "platforms" : "[A,G,W]",
        "tagNames" : "[TestTag...]",
        "userIds" : [ "MyUserId", ... ],
      },
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<send-message>
  <ArrayOfMessageBodyArray>
    <ArrayOfMessageBody>
      <message alert="New update available"/>
      <settings>
        <apns
          delayWhileIdle="true"
          payload=""
          sound="song.mp3"
          timeToLive="100"/>
        <gcm
          badge="1"
          iosActionKey="Ok"
          payload=""
          sound="song.mp3"
          type="SILENT"/>
        <wns
          badge=""
          cachePolicy="false"
          expirationTime="20"
          raw=""
          tile=""
          toast=""/>
      </settings>
      <target
        deviceIds="[TestDeviceId,..]"
        platforms="[A,G,W]"
        tagNames="[TestTag...]">
        <userIds>
          <userId>MyUserId</userId>
          ...
        </userIds>
      </target>
    </ArrayOfMessageBody>
    ...
  </ArrayOfMessageBodyArray>
</send-message>
```

Payload Properties

The payload has the following properties:

ArrayOfMessageBody
: The array of message

The *array of messages* has the following properties:

message
: The notification message to be sent

settings
: The settings for GCM, APNS and WNS.

target
: The targets for sending notification

The *alert messages* has the following properties:

alert
: A string to be displayed in the alert.

The *message settings* has the following properties:

apns
: Attributes for sending message to an iOS device

gcm
: Attributes for sending message to an Android device

wns
: Attributes for sending message to an Windows device

The *apns settings* 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 settings* has the following properties:

delayWhileIdle
: A Boolean value to indicate 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. The default value is 4 weeks, and must be set as a JSON number.

The *wns settings* has the following properties:

badge
: Optional. A numeric or string value that indicates a prdefined glyph to be displayed.

cachePolicy
: Optional. A boolean value that indicates if the notification should be cached or not.

expirationTime
: Optional. Expriry time of the notification.

raw
: Optional. A JSON block that is transferred to the application only if the application is already open.

tile
: Optional. Updates to tile to communicate new information to the user

toast
: Optional. Updates to the toast to communicate new information to the user

The *badge* has the following properties:

value
: Optional. A numeric or string value that indicates a prdefined glyph to be displayed.

version
: Optional. Version of the payload.

The *raw* has the following properties:

payload
: Optional. A JSON block that is transferred to the application only if the application is already open.

The *tile* has the following properties:

tag
: Optional. A string value that is set as label for the notification. Used in notification cycling.

visual
Optional. Visual settings for the notification

The *visual* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

baseUri
Optional. Base URI to be combined with the relative URIs.

binding
Optional. For tile notifications, its a JSON array containing JSON blocks of binding attributes. For toast notification, its a JSON block of binding attributes.

branding
Optional. Indicates whether logo or app's name to be shown. Default is None.

contentId
Optional. A string value that identifies the notification content. Only applies to tile notifications.

lang
Optional. Locale of the payload.

version
Optional. Version of the payload.

The *toast* has the following properties:

audio
Optional. Audio settings for the notification

duration
Optional. Notification will be displayed for the specified duration. Should be 'short' or 'long'.

launch
Optional. A string value that is passed to the application when it is launched by tapping or clicking the toast notification.

visual
Optional. Visual settings for the notification

The *audio* has the following properties:

loop
Optional. A boolean value to indicate if the sound should be repeated or not.

silent
Optional. A boolean value to indicate if the sound should be played or not.

src
Optional. A string value that specifies the notification sound type or path to local audio file.

The *target settings* has the following properties:

deviceIds
A JSON array of the device identifiers. Devices with these ids receive notification.

platforms
A JSON array of platforms. The devices that run on these platforms receive notification. Supported values are A (Apple/iOS), G (Google/Android) and W (Microsoft/Windows).

tagNames
A JSON array of tags. The devices that are subscribed to these tags receive notification.

userIds
An array of users represented by their userIds to send the notification. This is a unicast notification.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Send Message (POST)

Sends message with different options.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

`POST`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/messages

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

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

JSON Example

```
{
  "message" : {
    "alert" : "New update available",
  },
  "settings" : {
    "apns" : {
      "delayWhileIdle" : true,
      "payload" : "",
      "sound" : "song.mp3",
      "timeToLive" : 100,
    },
    "gcm" : {
      "badge" : 1,
      "iosActionKey" : "Ok",
      "payload" : "",
      "sound" : "song.mp3",
      "type" : "SILENT",
    },
    "wns" : {
      "badge" : ,
      "cachePolicy" : false,
      "expirationTime" : 20,
      "raw" : ,
      "tile" : ,
      "toast" : ,
    },
  },
  "target" : {
    "deviceIds" : "[TestDeviceId,..]",
    "platforms" : "[A,G,W]",
    "tagNames" : "[TestTag...]",
    "userIds" : [ "MyUserId", ... ],
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<send-message>
  <message alert="New update available"/>
  <settings>
    <apns
```

```
      delayWhileIdle="true"
      payload=""
      sound="song.mp3"
      timeToLive="100"/>
    <gcm
      badge="1"
      iosActionKey="Ok"
      payload=""
      sound="song.mp3"
      type="SILENT"/>
    <wns
      badge=""
      cachePolicy="false"
      expirationTime="20"
      raw=""
      tile=""
      toast=""/>
  </settings>
  <target
    deviceIds="[TestDeviceId,..]"
    platforms="[A,G,W]"
    tagNames="[TestTag...]">
    <userIds>
      <userId>MyUserId</userId>
      ...
    </userIds>
  </target>
</send-message>
```

Payload Properties

The payload has the following properties:

message
: The notification message to be sent

settings
: The settings for GCM, APNS and WNS

target
: The targets for sendig the notification

The *alert messages* has the following properties:

alert
: A string to be displayed in the alert.

The *message settings* has the following properties:

apns
: Attributes for sending message to an iOS device

gcm
: Attributes for sending message to an Android device

wns
: Attributes for sending message to an Windows device

The *apns settings* 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 settings* has the following properties:

delayWhileIdle
: A Boolean value to indicate 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. The default value is 4 weeks, and must be set as a JSON number.

The *wns settings* has the following properties:

badge
: Optional. A numeric or string value that indicates a prdefined glyph to be displayed.

cachePolicy
: Optional. A boolean value that indicates if the notification should be cached or not.

expirationTime
: Optional. Exprĭry time of the notification.

raw
: Optional. A JSON block that is transferred to the application only if the application is already open.

tile
: Optional. Updates to tile to communicate new information to the user

toast
: Optional. Updates to the toast to communicate new information to the user

The *badge* has the following properties:

value
: Optional. A numeric or string value that indicates a prdefined glyph to be displayed.

version
: Optional. Version of the payload.

The *raw* has the following properties:

payload
: Optional. A JSON block that is transferred to the application only if the application is already open.

The *tile* has the following properties:

tag
: Optional. A string value that is set as label for the notification. Used in notification cycling.

visual
: Optional. Visual settings for the notification

The *visual* has the following properties:

addImageQuery
: Optional. A boolean value that indicates if the query string need to be appended to image URI.

baseUri
: Optional. Base URI to be combined with the relative URIs.

binding
: Optional. For tile notifications, its a JSON array containing JSON blocks of binding attributes. For toast notification, its a JSON block of binding attributes.

branding
: Optional. Indicates whether logo or app's name to be shown. Default is None.

contentId
: Optional. A string value that identifies the notification content. Only applies to tile notifications.

lang
: Optional. Locale of the payload.

version
: Optional. Version of the payload.

The *toast* has the following properties:

audio
: Optional. Audio settings for the notification

duration
: Optional. Notification will be displayed for the specified duration. Should be 'short' or 'long'.

launch
: Optional. A string value that is passed to the application when it is launched by tapping or clicking the toast notification.

visual
: Optional. Visual settings for the notification

The *audio* has the following properties:

`loop`
Optional. A boolean value to indicate if the sound should be repeated or not.

`silent`
Optional. A boolean value to indicate if the sound should be played or not.

`src`
Optional. A string value that specifies the notification sound type or path to local audio file.

The *target settings* has the following properties:

`deviceIds`
A JSON array of the device identifiers. Devices with these ids receive notification.

`platforms`
A JSON array of platforms. The devices that run on these platforms receive notification. Supported values are A (Apple/iOS), G (Google/Android) and W (Microsoft/Windows).

`tagNames`
A JSON array of tags. The devices that are subscribed to these tags receive notification.

`userIds`
An array of users represented by their userIds to send the notification. This is a unicast notification.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

Transaction (GET)

Retrieves information about a specific transaction.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpmonitor`**
- **`mfpoperator`**

Method

GET

Path

/management-apis/2.0/runtimes/*runtime-name*/transactions/*transaction-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/transactions/1?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.

`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" : "8.0",
  "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="8.0"
  status="FAILURE"
  timeCreated="2014-04-13T00:18:36.979Z"
  timeUpdated="2014-04-14T00:18:36.979Z"
  type="DELETE_ADAPTER"
  userName="demouser">
  <description/>
  <errors>
    <error details="An internal error occured."/>
    ...
  </errors>
  <project name="myproject"/>
</transaction>
```

Response Properties

The response has the following properties:

appServerId
The id of the web application server.

description
The details of the transaction, depending on the transaction type.

errors
The errors occured during the transacton.

id The id of the transaction.

productVersion
The exact product version.

project
The current project.

status
The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
The date in ISO 8601 format when the adapter was created.

timeUpdated
The date in ISO 8601 format when the adapter was updated.

type
The type of the transaction.

userName
The user that initiated the transaction.

The *error* has the following properties:

details
The main error message.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or the transaction is not found.`

500

`An internal error occurred.`

Transactions (GET)

Retrieves information of all transactions.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpmonitor**
- **mfpoperator**

Method

```
GET
```

Path

/management-apis/2.0/runtimes/*runtime-name*/transactions

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/transactions?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.

file
If this parameter is set to `true`, the transactions are delivered as a compressed file (.zip). In this case, paging and mode parameters are ignored.

locale
The locale used for error messages.

mode
: If this parameter is set to `errors`, only erroneous transactions are listed. Ootherwise, all transactions are listed.

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

Details about 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" : "8.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<transactions
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="8.0"
  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 transactions

nextPageBookmark
: The bookmark of the next page if only one page of transactions is returned.

pageNumber
: The page index if only one page of transactions is returned.

pageSize
: The page size if only one page of transactions is returned.

prevPageBookmark
: The bookmark of the previous page if only one page of transactions is returned.

productVersion
: The exact product version.

startIndex
: The start index in the total list if only one 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.`

Remove Subscription (DELETE)

Unsubscribes the specified device from a tag.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**

Method

`DELETE`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

Query Parameters

Query parameters are optional.

`deviceId`
The unique ID for the device

`locale`
The locale used for error messages.

`tagName`
The name of the tag to unsubscribe from

Produces

application/json, application/xml, text/xml

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Update Device Registration (PUT)

Updates push device registration with the new user ID or the specified token. In most use cases, only the user ID is updated.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpoperator`**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/devices/*device-id*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

`mfpPushEnableBroadcast`
Participate in the broadcast messaging.

Consumes

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "deviceId" : "JeremyiOSPhone",
  "platform" : "A",
  "token" : "c6a41224 23333917 9fde1532",
  "userId" : "Jeremy",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<device-update
  deviceId="JeremyiOSPhone"
  platform="A"
  token="c6a41224 23333917 9fde1532"
  userId="Jeremy"/>
```

Payload Properties

The payload has the following properties:

deviceId
The unique identifier of the device

platform
The device platform

token
The unique push token of the device

userId
The identifier of the user of the device.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Update APNs settings (PUT)

Uploads an APNs certificate to the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**

- **mfpdeployer**
- **mfpoperator**

Method

PUT

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/apnsConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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

multipart/form-data

Produces

application/json, application/xml, text/xml

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

415

```
Unsupported Media Type - The content type specified in Content-Type header is not application/json.
```

500

```
An internal error occurred.
```

Update GCM settings (PUT)

Uploads a GCM certificate to the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpoperator`**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/gcmConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/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.

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

JSON Example

```
{
  "apiKey" : "AIzaSyBnWWReKAFrOPiw75QQAcRM",
  "senderId" : "11639055112",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushGCM
  apiKey="AIzaSyBnWWReKAFrOPiw75QQAcRM"
  senderId="11639055112"/>
```

Payload Properties

The payload has the following properties:

apiKey
GCM Api Key

senderId
The project ID that is signed up at Google API console

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

415

```
Unsupported Media Type - The content type specified in Content-Type header is not application/json.
```

500

```
An internal error occurred.
```

Update WNS Settings (PUT)

Uploads an WNS certificate to the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **mfpadmin**
- **mfpdeployer**
- **mfpoperator**

Method

```
PUT
```

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/wnsConf

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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

JSON Example

```
{
  "clientSecret" : "712345dummyvalues12345",
  "packageSID" : "ms-app://s-1-15-2-dummyvalues12345",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushWNS
  clientSecret="712345dummyvalues12345"
  packageSID="ms-app://s-1-15-2-dummyvalues12345"/>
```

Payload Properties

The payload has the following properties:

`clientSecret`
The Secret Key

`packageSID`
Package Security Identifier (SID)

Errors

400

```
The request was not understood by the push server.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or application is not found or not running.
```

415

`Unsupported Media Type - The content type specified in Content-Type header is not application/json.`

500

`An internal error occurred.`

Update Tag Information (PUT)

Updates the tag that is idenfitied by the `tagName` parameter for the application referenced by the application name.

Roles

Users in the following roles are authorized to perform this operation:

- **`mfpadmin`**
- **`mfpdeployer`**
- **`mfpoperator`**

Method

`PUT`

Path

/management-apis/2.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags/*tag-name*

Example

```
https://www.example.com/mfpadmin/management-apis/2.0/runtimes/myruntime/notifications/applications/my
```

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.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "description" : "This is a sample of a modified tag.",
  "name" : "SampleTag",
}
```

Payload Properties

The payload has the following properties:

`description`
The description of the tag to be modified.

`name`
The name of the tag to be modified.

Errors

400

`The request was not understood by the push server.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or application is not found or not running.`

500

`An internal error occurred.`

REST API for the MobileFirst Server push service

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 from the push service

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 registration.

Method

`DELETE`

Path

/apps/*applicationId*/devices/*deviceId*

Example

`https://example.com:443/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 "devices.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

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 an existing device registration of push

Description

Device registrations for a push service that are retrieved for a specific deviceId.

Method

`GET`

Path

/apps/*applicationId*/devices/*deviceId*

Example

```
https://example.com:443/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 "devices.read" and "push.application.<*applicationId*>" 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 device registration that is retrieved.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-05-20T11:42:11Z",
  "deviceId" : "12345-6789",
  "lastUpdatedTime" : "2015-05-20T11:42:11Z",
  "phoneNumber" : "123456789",
  "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.

phoneNumber
: Phone number to be used for SMS based notification.

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.`

Push Device Registration (POST)

Creates a device registration with the push service.

Description

The device registrations happens from the device. The deviceId is the unique ID for the device for the application.

Method

`POST`

Path

/apps/*applicationId*/devices

Example

`https://example.com:443/imfpush/v1/apps/myapp/devices`

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 "devices.write" and "push.application.<*applicationId*>" 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.

Consumes

application/json

Produces

application/json

Payload

The details of the device registration.

JSON Example

```
{
  "deviceId" : "12345-6789",
  "phoneNumber" : "123456789",
  "platform" : "A",
  "token" : "xyz",
}
```

Payload Properties

The payload has the following properties:

deviceId
Unique id of the device.

phoneNumber
Phone number to be used for SMS based notification.

platform
The device platform. 'A' refers to Apple(iOS) devices, 'G' refers to Google(Android) and 'W' refers to Microsoft(Windows) devices

token
Device token obtained via the service provider

Response

The details of the application.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-05-20T11:42:11Z",
  "deviceId" : "12345-6789",
  "lastUpdatedTime" : "2015-05-20T11:42:11Z",
  "phoneNumber" : "123456789",
  "platform" : "A",
  "token" : "xyz",
  "userAgent" : "TestUserAgent",
  "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
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.

phoneNumber
Phone number to be used for SMS based notification.

platform
The device platform. 'A' refers to Apple(iOS) devices, 'G' refers to Google(Android) and 'W' refers to Microsoft(Windows) devices

token
Device token obtained via the service provider

userAgent
The user agent for the the device registration

userId
The user identifier for the the device registration

Errors

400

```
Bad Request - The request was not understood by the push server. An invalid JSON could result in thi
```

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

405

```
Unsupported Content type - The content type specified in Content-Type header is not application/json
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Push Device Registrations (GET)

Retrieves all or a subset of existing device registration(s) of push.

Description

Device registrations for the push service are retrieved for the specified criteria

Method

`GET`

Path

/apps/*applicationId*/devices

Example

```
https://example.com:443/imfpush/v1/apps/myapp/devices?expand=true&filter=platform==A&offset=0&size
```

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`
Search criteria filter. 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 device registrations only for the specified user.

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 "devices.read" and "push.application.<*applicationId*>" 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 device registration that is retrieved.

JSON Example

```
{
  "devices" : [
    {
      "deviceId" : "12345-6789",
      "phoneNumber" : "123456789",
      "platform" : "A",
      "token" : "xyz",
      "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.

phoneNumber
Phone number to be used for SMS based notification.

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 Registration (PUT)

Updates an existing device registration for the push service.

Description

The push device registration is updated with the new user ID 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/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 "devices.write" and "push.application.<*applicationId*>" 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.

Consumes

application/json

Produces

application/json

Payload

The details of the device registration will be updated.

JSON Example

```
{
  "deviceId" : "12345-6789",
  "phoneNumber" : "123456789",
  "token" : "xyz",
  "userId" : "admin",
}
```

Payload Properties

The payload has the following properties:

deviceId
: The unique id of the device.

phoneNumber
: Phone number to be used for SMS based notification.

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",
  "phoneNumber" : "123456789",
  "platform" : "A",
  "token" : "xyz",
  "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.

phoneNumber
Phone number to be used for SMS based notification.

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.`

Push Device Subscription (DELETE)

Delete subscription by subscriptionId.

Description

Using the subscriptionId it unsubscribes the tag from the device. The call would not delete the device registration or the tag.

Method

`DELETE`

Path

/apps/*applicationId*/subscriptions/*subscriptionId*

Example

`https://example.com:443/imfpush/v1/apps/myapp/subscriptions/mysubscription`

Path Parameters

applicationId
The name or identifier of the application.

subscriptionId
The identifier of the subscription.

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 "subscriptions.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*. This parameter is mandatory.

Produces

application/json

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The subscription with the specified subscriptionId is not found.`

500

`An internal error occurred.`

Push Device Subscription (GET)

Retrieves an existing subscription of push.

Description

The subscription referenced by the subscriptionId is retrieved.

Method

`GET`

Path

/apps/*applicationId*/subscriptions/*subscriptionId*

Example

`https://example.com:443/imfpush/v1/apps/myapp/subscriptions/mysubscription`

Path Parameters

applicationId
The name or identifier of the application.

subscriptionId
The identifier of the subscription.

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 "subscriptions.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.This parameter is mandatory.

Produces

application/json

Response

The details of the device subscription 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.`

404

`The subscription with the specified subscriptionId is not found.`

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/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 "subscriptions.write" and "push.application.<*applicationId*>" 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.

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 subscription 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/json`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Device Subscriptions (GET)

Retrieves all or a subset of existing subscriptions

Description

Retrieves subscriptions for the push service for the specified criteria.

Method

`GET`

Path

/apps/*applicationId*/subscriptions

Example

```
https://example.com:443/imfpush/v1/apps/myapp/subscriptions?deviceId=12345-6789&expand=true&filter=ta
```

Path Parameters

`applicationId`
The name or identifier of the application

Query Parameters

Query parameters are optional.

`deviceId`
Retrieves subscriptions only for the specified deviceId

`expand`
Retrieves 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
Retrieves 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 "subscriptions.read" and "push.application.<*applicationId*>" 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 device subscription 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 subscriptions.

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 Tags (GET)

Retrieves all tags of Push

Method

`GET`

Path

/apps/*applicationId*/tags

Example

```
https://example.com:443/imfpush/v1/apps/myapp/tags?expand=true&filter=platform==A&offset=0&size=10&s
```

Path Parameters

`applicationId`
The name or identifier of the application

Query Parameters

Query parameters are optional.

`expand`
Retrieves detailed information about applications.

`filter`
Search criteria filter. 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.

`subscriptionCount`
Retrieves the number of tag subscriptions

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 "tags.read" and "push.application.<*applicationId*>" 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

```
{
  "pageInfo" : {
    "count" : "2",
    "next" : "",
    "previous" : "",
    "totalCount" : "10",
  },
  "tags" : [
    {
      "createdMode" : "API",
      "createdTime" : "2015-08-22T18:19:58Z",
      "description" : "Description about SampleTag",
      "href" : "https://example.com:443/imfpush/v1/apps/testApp/tags/SampleTag",
      "lastUpdatedTime" : "2015-08-22T18:19:58Z",
      "name" : "SampleTag",
    },
    ...
  ],
}
```

Response Properties

The response has the following properties:

pageInfo
: The pagination information

tags
: The array of applications.

The *pageInfo* has the following properties:

count
: The number of tags that are retrieved

next
: A hyperlink to the next page

previous
: A hyperlink to the previous page

totalCount
: The total number of application tags present for the given search criteria

The *tagnames* has the following properties:

createdMode
: Defaults to API

createdTime
: The time at which the tag was created

description
: The description of the tag

href
: The URL to the tag

lastUpdatedTime
: The time at which the tag was last updated

name
: An unique name of the tag in the application

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`A tag with the specified name is not found.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Applications (GET)

Retrieves all the applications.

Method

`GET`

Path

/apps/

Example

`https://example.com:443/imfpush/v1/apps/?expand=true&filter=platform==A&offset=0&size=10`

Query Parameters

Query parameters are optional.

`expand`
Retrieves detailed information about applications.

`filter`
Search criteria filter. 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.

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 "apps.read" and "push.application.*" 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.

Produces

application/json

Response

The details of all the applications.

JSON Example

```
{
  "applications" : [
    {
      "applicationId" : "testApp",
    },
    ...
  ],
  "pageInfo" : {
    "count" : "2",
    "next" : "",
    "previous" : "",
    "totalCount" : "10",
  },
}
```

Response Properties

The response has the following properties:

applications
The array of applications.

pageInfo
The pagination information

The *applications* has the following properties:

applicationId
The applicationId.

The *pageInfo* has the following properties:

count
The number of applications that are retrieved

next
A hyperlink to the next page

previous
A hyperlink to the previous page

totalCount
The total number of applications 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 Application (POST)

Creates a new server application for the push service.

Description

The applicationId is an unique application ID for this application. Application is a parent resource for devices, subscriptions, tags and messages. The application must be created before accessing any of the child resources. If the application is deleted, all the children are deleted. The application holds the configurations, such as the

Apple Push Notification Service (APNS) and Google Cloud Message (GCM) configuration, which is required by the push service to send messages. The API first creates the application and then sets the APNS and GCM settings.

Method

`POST`

Path

/apps/

Example

```
https://example.com:443/imfpush/v1/apps/
```

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 "apps.write" and "push.application.<*applicationId*>" 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.

Produces

application/json

Payload

The details of the application.

JSON Example

```
{
  "applicationId" : "testApp",
  "enabled" : "true",
}
```

Payload Properties

The payload has the following properties:

`applicationId`
: The application Id.

`enabled`
: Optinal. The status of the applicaton. Default is true

Response

The details of the application.

JSON Example

```
{
  "applicationId" : "testApp",
  "enabled" : "true",
}
```

Response Properties

The response has the following properties:

applicationId
The application Id.

enabled
The status of the application.

Errors

400

`Bad Request - The request was not understood by the push server. An invalid JSON could result in thi`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

405

`Unsupported Content type - The content type specified in Content-Type header is not application/json`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Application (GET)

Retrieves the application, which is referenced by the applicationId parameter.

Method

`GET`

Path

/apps/*applicationId*/status

Example

`https://example.com:443/imfpush/v1/apps/myapp/status`

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 "apps.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

The status of the application.

JSON Example

```
{
  "enabled" : "true",
}
```

Response Properties

The response has the following properties:

enabled
The status of the application.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

500

`An internal error occurred.`

Push Application (DELETE)

Deletes an application, which is referenced by the applicationId parameter.

Description

After the application is deleted the tags, devices, subscriptions and message resources associated with the application are also deleted.

Method

`DELETE`

Path

/apps/*applicationId*

Example

`https://example.com:443/imfpush/v1/apps/myapp`

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 "apps.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

500

`An internal error occurred.`

Push Application Settings (GET)

Retrieves appplication settings

Description

This can be used to find the mode of the application.

Method

`GET`

Path

/apps/*applicationId*/settings

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings`

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 "settings.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

Retrieve application settings.

JSON Example

```
{
  "apnsConf" : "https://example.com:443/imfpush/v1/apps/testApp/settings/apnsConf",
  "applicationId" : "testApp",
  "gcmConf" : "https://example.com:443/imfpush/v1/apps/testApp/settings/gcmConf",
  "mode" : "PRODUCTION",
}
```

Response Properties

The response has the following properties:

apnsConf
: Reference link to APNS settings.

applicationId
: The application Id.

gcmConf
: Reference link to GCM settings.

mode
: The operation mode

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push APNS Settings (GET)

Retrieves APNS settings for the application

Method

GET

Path

/apps/*applicationId*/settings/apnsConf

Example

```
https://example.com:443/imfpush/v1/apps/myapp/settings/apnsConf
```

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 "apnsConf.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

Retrieves APNS settings for the application.

JSON Example

```
{
  "certificate" : "apns-certificate.p12",
  "isSandBox" : "true",
  "validUntil" : "2016-09-06T05:51:11.000Z",
}
```

Response Properties

The response has the following properties:

certificate
The name of the certificate.

isSandBox
The certificate type.

validUntil
The certificate validity date.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push APNS settings (PUT)

Uploads an APNS certificate to the application referenced by the applicationId

Method

`PUT`

Path

/apps/*applicationId*/settings/apnsConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/apnsConf`

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 "apnsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Content-Type
Specify the content type. For example: multipart/form-data. This parameter has to be mandatorily set.

Consumes

multipart/form-data

Produces

application/json

Form-data Parameters

`certificate`
(file) The APNS certificate.

`password`
(String) Password for the APNS certificate

`isSandBox`
(boolean) The APNS certificate type.

Response

Successfully updated the APNS settings.

JSON Example

```
{
  "certificate" : "apns-certificate.p12",
  "isSandBox" : "true",
  "validUntil" : "2016-09-06T05:51:11.000Z",
}
```

Response Properties

The response has the following properties:

`certificate`
The name of the APNS certificate.

`isSandBox`
The APNS certificate type.

`validUntil`
The APNS certificate validity date.

Errors

400

`Bad Request -  The request was not understood by the push server. An invalid data in the input.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push APNS settings (DELETE)

Deletes APNS settings for the application

Method

`DELETE`

Path

/apps/*applicationId*/settings/apnsConf

Example

```
https://example.com:443/imfpush/v1/apps/myapp/settings/apnsConf
```

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 "apnsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
The application does not exist.
```

500

```
An internal error occurred.
```

Push GCM Settings (GET)

Retrieves GCM settings for the application

Method

```
GET
```

Path

/apps/*applicationId*/settings/gcmConf

Example

```
https://example.com:443/imfpush/v1/apps/myapp/settings/gcmConf
```

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 "gcmConf.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

Retrieves GCM settings for the application.

JSON Example

```
{
  "apiKey" : "AxBNGYUwehjokn",
  "senderId" : "123456789",
}
```

Response Properties

The response has the following properties:

apiKey
The GCM API Key.

senderId
The GCM SenderId.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push GCM Settings (PUT)

Updates GCM settings referenced by the applicationId

Method

PUT

Path

/apps/*applicationId*/settings/gcmConf

Example

```
https://example.com:443/imfpush/v1/apps/myapp/settings/gcmConf
```

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 "apnsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Content-Type
Specify the content type. For example: application/json. This parameter has to be mandatorily set.

Consumes

application/json

Produces

application/json

Payload

The details of the gcm settings.

JSON Example

```
{
  "apiKey" : "AxBNGYUwehjokn",
  "senderId" : "123456789",
}
```

Payload Properties

The payload has the following properties:

apiKey
The GCM API Key.

senderId
The GCM SenderId.

Response

Retrieves GCM settings for the application.

JSON Example

```
{
  "apiKey" : "AxBNGYUwehjokn",
  "senderId" : "123456789",
}
```

Response Properties

The response has the following properties:

apiKey
: The GCM API Key.

senderId
: The GCM SenderId.

Errors

400

`Bad Request - The request was not understood by the push server. An invalid data in the input.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push GCM Settings (DELETE)

Deletes GCM settings for the application

Method

`DELETE`

Path

/apps/*applicationId*/settings/gcmConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/gcmConf`

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 "gcmConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

500

`An internal error occurred.`

Push WNS Settings (GET)

Retrieves WNS settings for the application

Method

`GET`

Path

/apps/*applicationId*/settings/wnsConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/wnsConf`

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 "wnsConf.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

Retrieves WNS settings for the application.

JSON Example

```
{
  "clientSecret" : "Vex8L9WOFZuj95euaLrvSH7XyoDhLJc7",
  "packageSID" : "ms-app://S-1-15-2-2972962901-2322836549-3722629029-1345238579-3987825745-215561607
}
```

Response Properties

The response has the following properties:

clientSecret
The Client secret.

packageSID
Package SID

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push WNS Settings (PUT)

Updates WNS settings referenced by the applicationId

Method

`PUT`

Path

/apps/*applicationId*/settings/wnsConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/wnsConf`

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 "wnsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Content-Type
: Specify the content type. For example: application/json. This parameter has to be mandatorily set.

Consumes

application/json

Produces

application/json

Payload

The details of the wns settings.

JSON Example

```
{
  "clientSecret" : "Vex8L9WOFZuj95euaLrvSH7XyoDhLJc7",
  "packageSID" : "ms-app://S-1-15-2-2972962901-2322836549-3722629029-1345238579-3987825745-215561
}
```

Payload Properties

The payload has the following properties:

clientSecret
: The Client secret.

packageSID
: Package SID

Response

Successfully updated the WNS settings.

JSON Example

```
{
  "clientSecret" : "Vex8L9WOFZuj95euaLrvSH7XyoDhLJc7",
  "packageSID" : "ms-app://S-1-15-2-2972962901-2322836549-3722629029-1345238579-3987825745-215561
}
```

Response Properties

The response has the following properties:

clientSecret
: The Client secret.

packageSID
: Package SID

Errors

400

`Bad Request - The request was not understood by the push server. An invalid data in the input.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push WNS settings (DELETE)

Deletes WNS settings for the application

Method

`DELETE`

Path

/apps/*applicationId*/settings/wnsConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/wnsConf`

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 "wnsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

500

`An internal error occurred.`

Push Application (PUT)

Updates an existing application status. The API can be used to enable or disable an application.

Method

`PUT`

Path

/apps/*applicationId*/status

Example

```
https://example.com:443/imfpush/v1/apps/myapp/status
```

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 "apps.write" and "push.application.<*applicationId*>" 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.

Consumes

application/json

Produces

application/json

Payload

The details of the application.

JSON Example

```
{
  "enabled" : "true",
}
```

Payload Properties

The payload has the following properties:

`enabled`
: The status of the application

Response

The details of the application.

JSON Example

```
{
  "applicationId" : "testApp",
  "enabled" : "true",
}
```

Response Properties

The response has the following properties:

`applicationId`
: The application Id.

`enabled`
: The status of the application.

Errors

400

```
Bad Request - The request was not understood by the push server. An invalid JSON could result in thi
```

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
The applicationId does not exist.
```

405

```
Unsupported Content type - The content type specified in Content-Type header is not application/json
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Push 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/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 "messages.write" and "push.application.<*applicationId*>" 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",
  },
  "notificationType" : 1,
  "settings" : {
    "apns" : {
      "badge" : 1,
      "category" : 1,
      "iosActionKey" : "Ok",
      "payload" : {"custom":"data"},
      "sound" : "song.mp3",
      "type" : "SILENT",
    },
    "gcm" : {
      "bridge" : false,
      "category" : "email",
      "collapseKey" : "testkey",
      "delayWhileIdle" : false,
      "payload" : {"custom":"data"},
      "priority" : "low",
      "redact" : "Test Redact Message",
      "sound" : "song.mp3",
      "sync" : false,
      "timeToLive" : 10,
      "visibility" : "public",
    },
    "wns" : {
      "badge" : {"value":"10"},
      "cachePolicy" : false,
      "expirationTime" : 20,
      "raw" : {"payload":{"custom":"data"}},
      "tile" : {"visual":{"binding":[{"template":"TileSquareText04", "text": [{"content":"Text1"}]},
      "toast" : {"launch":{"custom":"data"}, "visual":{"binding":{"template":"ToastText04","text":[{
    },
  },
  "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

`notificationType`
Integer value to indicate the channel (Push/SMS) used to send message. Allowed values are 1 (only Push), 2 (only SMS) and 3 (Push and SMS)

`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.

`wns`
Attributes for sending message to a windows device.

The *apns* has the following properties:

`badge`
An integer value to be displayed in a badge on the application icon.

category
: Name of the category for iOS8 interactive push notifications.

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:

bridge
: A Boolean value that indicates whether the notification should be bridged or not to other devices connected to this handheld device. Only applies to Android 5.0 or higher.

category
: A string value that indicates the category to which this notification belongs. Allowed values are 'call', 'alarm', 'email', 'err', 'event', 'msg', 'progress', 'promo', 'recommendation', 'service', 'social', 'status', and 'transport'. Only applies to Android 5.0 or higher.

collapseKey
: A string value that indicates that the message can be replaced. When multiple messages are queued up in GCM Servers with the same key, only the last one is delivered.

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. Default value is false

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.

priority
: A string value that indicates the priority of this notification. Allowed values are 'max', 'high', 'default', 'low' and 'min'. High/Max priority notifications along with 'sound' field may be used for Heads up notification in Android 5.0 or higher.

redact
: A string to be displayed in the alert as a redacted version of the original content when the visibility level is 'private'. Only applies to Android 5.0 or higher.

sound
: The name of a sound file on the device to play when the notification arrives to the device.

sync
A Boolean value that indicates whether the notification should be sync'd between devices of the same user, that is, if a notification is handled on a device it gets dismissed on the other devices of the same user

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.

visibility
A string value that indicates the visibility level of notification content on the secured lock screen in Android L devices. Allowed values are 'public, 'private' and 'secret'. Only applies to Android 5.0 or higher.

The *wns* has the following properties:

badge

cachePolicy
A boolean value that indicates if the notification should be cached or not.

expirationTime
Optional. Exprіry time of the notification.

raw

tile

toast

The *badge* has the following properties:

value
Optional. A numeric or string value that indicates a prdefined glyph to be displayed.

version
Optional. Version of the payload.

The *raw* has the following properties:

payload
Optional. A JSON block that is transferred to the application only if the application is already open.

The *tile* has the following properties:

tag
Optional. A string value that is set as label for the notification. Used in notification cycling.

visual

The *visual* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

baseUri
Optional. Base URI to be combined with the relative URIs.

binding
For tile notifications, its a JSON array containing JSON blocks of binding attributes. For toast notification, its a JSON block of binding attributes.

branding
Optional. Indicates whether logo or app's name to be shown. Default is None.

contentId
Optional. A string value that identifies the notification content. Only applies to tile notifications.

lang
Optional. Locale of the payload.

version
Optional. Version of the payload.

The *binding* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

baseUri
Optional. Base URI to be combined with the relative URIs.

branding
Optional. Indicates whether logo or app's name to be shown. Default is None.

contentId
Optional. A string value that identifies the notification content. Only applies to tile notifications.

fallback
Optional. Template to be used as a fallback.

image
Optional. A JSON array containing JSON blocks of following image attributes.

lang
Optional. Locale of the payload.

template
Mandatory. Template type of the notification.

text
Optional. A JSON array containing JSON blocks of following text attributes.

The *image* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

alt
Optional. Image description.

src
Mandatory. Image URI.

The *text* has the following properties:

content
Mandatory. A string value that is displayed in the toast.

lang
Optional. Locale of the payload.

The *toast* has the following properties:

audio

duration
Optional. Notification will be displayed for the specified duration. Should be 'short' or 'long'.

launch
Optional. A string value that is passed to the application when it is launched by tapping or clicking the toast notification.

visual

The *audio* has the following properties:

loop
Optional. A boolean value to indicate if the sound should be repeated or not.

silent
Optional. A boolean value to indicate if the sound should be played or not.

src
Optional. A string value that specifies the notification sound type or path to local audio file.

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 W (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.

Response

The details of the message that is retrieved.

JSON Example

```
{
  "message" : {
    "message" : {
      "alert" : "TestMessage",
    },
  },
  "messageId" : "1234",
}
```

Response Properties

The response has the following properties:

`message`
: The array of messages to be sent

`messageId`
: The unique identifier of the message.

The *messages* has the following properties:

`message`
: The message to be sent

The *message* has the following properties:

`alert`
: The message text.

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.`

Push Message (GET)

Retrieves the message details by messageId.

Description

Requires the following JNDI properties to be set for MobileFirst Push:
- `imfpush/mfp.push.messages.persist.size` (Queue size to store messages before writing to database).
- `imfpush/mfp.push.messages.persist.delay.mins` (Delay in minutes to write sent messages in database).

Method

`GET`

Path

/apps/*applicationId*/messages/*messageId*

Example

`https://example.com:443/imfpush/v1/apps/myapp/messages/mymessage`

Path Parameters

applicationId
The name or identifier of the application.

messageId
The identifier of the message.

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 `messages.read` and `push.application.<*applicationId*>` obtained using the confidential client in the format bearer *token*. This is a mandatory parameter.

Produces

`application/json`

Response

The details of the message that is retrieved.

JSON Example

```
{
  "alert" : "TestMessage",
  "messageId" : "1234",
}
```

Response Properties

The response has the following properties:

alert
The message text.

messageId
The unique identifier of the message.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The message with the specified messageId is not found.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Message (DELETE)

Deletes the message details by messageId

Method

```
DELETE
```

Path

/apps/*applicationId*/messages/*messageId*

Example

```
https://example.com:443/imfpush/v1/apps/myapp/messages/mymessage
```

Path Parameters

`applicationId`
The name or identifier of the application

`messageId`
The identifier of the message

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 "messages.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
The message with the specified messageId is not found.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Push SMS Settings (GET)

Retrieves SMS settings for the application

Method

```
GET
```

Path

/apps/*applicationId*/settings/smsConf

Example

```
https://example.com:443/imfpush/v1/apps/myapp/settings/smsConf
```

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 "smsConf.read" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

Retrieves SMS settings for the application.

JSON Example

```
{
  "host" : "xyz.com",
  "name" : "TestGateway",
  "parameters" : [
    {
      "encode" : "true",
      "name" : "TestKey",
      "value" : "TestValue",
    },
    ...
  ],
  "port" : "80",
  "programName" : "/sendsms",
}
```

Response Properties

The response has the following properties:

host
The host name of the SMS Gateway

name
The name of the SMS Gateway

parameters
The array of parameters

port
The port number of the SMS Gateway

programName
The path of the SMS Gateway

The *parametersArray* has the following properties:

encode
The parameter should be encoded or not

name
The name of the parameter

value
The value of the parameter

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push SMS Settings (PUT)

Updates SMS settings referenced by the applicationId

Method

`PUT`

Path

/apps/*applicationId*/settings/smsConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/smsConf`

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 "smsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Content-Type
Specify the content type. For example: application/json. This parameter has to be mandatorily set.

Consumes

application/json

Produces

application/json

Payload

The details of the sms settings.

JSON Example

```
{
  "host" : "xyz.com",
  "name" : "TestGateway",
  "parameters" : [
    {
      "encode" : "true",
      "name" : "TestKey",
      "value" : "TestValue",
    },
    ...
  ],
  "port" : "80",
  "programName" : "/sendsms",
}
```

Payload Properties

The payload has the following properties:

host
The host name of the SMS Gateway

name
The name of the SMS Gateway

parameters
The array of parameters

port
The port number of the SMS Gateway

programName
The path of the SMS Gateway

The *parametersArray* has the following properties:

encode
The parameter should be encoded or not

name
The name of the parameter

value
The value of the parameter

Response

Successfully updated the SMS settings.

JSON Example

```
{
  "host" : "xyz.com",
  "name" : "TestGateway",
  "parameters" : [
    {
      "encode" : "true",
      "name" : "TestKey",
      "value" : "TestValue",
    },
    ...
  ],
  "port" : "80",
  "programName" : "/sendsms",
}
```

Response Properties

The response has the following properties:

host
The host name of the SMS Gateway

name
The name of the SMS Gateway

parameters
The array of parameters

port
The port number of the SMS Gateway

programName
The path of the SMS Gateway

The *parametersArray* has the following properties:

encode
The parameter should be encoded or not

name
The name of the parameter

value
The value of the parameter

Errors

400

`Bad Request - The request was not understood by the push server. An invalid data in the input.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push SMS settings (DELETE)

Deletes SMS settings for the application

Method

`DELETE`

Path

/apps/*applicationId*/settings/smsConf

Example

`https://example.com:443/imfpush/v1/apps/myapp/settings/smsConf`

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 "smsConf.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

500

`An internal error occurred.`

Push Tags (GET)

Retrieves all tags of Push

Method

GET

Path

/apps/*applicationId*/tags

Example

```
https://example.com:443/imfpush/v1/apps/myapp/tags?expand=true&filter=platform==A&offset=0&size=1
```

Path Parameters

applicationId
: The name or identifier of the application

Query Parameters

Query parameters are optional.

expand
: Retrieves detailed information about applications.

filter
: Search criteria filter. 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.

subscriptionCount
: Retrieves the number of tag subscriptions

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 "tags.read" and "push.application.<*applicationId*>" 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

```
{
  "pageInfo" : {
    "count" : "2",
    "next" : "",
    "previous" : "",
    "totalCount" : "10",
  },
  "tags" : [
    {
      "createdMode" : "API",
      "createdTime" : "2015-08-22T18:19:58Z",
      "description" : "Description about SampleTag",
      "href" : "https://example.com:443/imfpush/v1/apps/testApp/tags/SampleTag",
      "lastUpdatedTime" : "2015-08-22T18:19:58Z",
      "name" : "SampleTag",
    },
    ...
  ],
}
```

Response Properties

The response has the following properties:

`pageInfo`
The pagination information

`tags`
The array of applications.

The *pageInfo* has the following properties:

`count`
The number of tags that are retrieved

`next`
A hyperlink to the next page

`previous`
A hyperlink to the previous page

`totalCount`
The total number of application tags present for the given search criteria

The *tagnames* has the following properties:

`createdMode`
Defaults to API

`createdTime`
The time at which the tag was created

`description`
The description of the tag

`href`
The URL to the tag

`lastUpdatedTime`
The time at which the tag was last updated

`name`
An unique name of the tag in the application

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`A tag with the specified name is not found.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Tag (POST)

Creates a tag.

Description

Creates a tag with the unique name in the application, which is referenced by the applicationId parameter. The tag has associated with a description about the tag. The tag name cannot be updated after it is created

Method

`POST`

Path

/apps/*applicationId*/tags

Example

`https://example.com:443/imfpush/v1/apps/myapp/tags`

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 "tags.write" and "push.application.<*applicationId*>" 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.

Consumes

application/json

Produces

application/json

Payload

The details of the tag.

JSON Example

```
{
  "description" : "Description about SampleTag",
  "name" : "SampleTag",
}
```

Payload Properties

The payload has the following properties:

description
The description of the tag

name
An unique name of the tag in the application

Response

The details of the tag.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-08-22T18:19:58Z",
  "description" : "Description about SampleTag",
  "href" : "https://example.com:443/imfpush/v1/apps/testApp/tags/SampleTag",
  "lastUpdatedTime" : "2015-08-22T18:19:58Z",
  "name" : "SampleTag",
}
```

Response Properties

The response has the following properties:

createdMode
Defaults to API

createdTime
The time at which the tag was created

description
The description of the tag

href
The URL to the tag

lastUpdatedTime
The time at which the tag was last updated

name
An unique name of the tag in the application

Errors

400

```
Bad Request - The request was not understood by the push server. An invalid JSON could result in thi
```

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

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 Tag (GET)

Retrieves an existing tag of Push

Method

`GET`

Path

/apps/*applicationId*/tags/*tagName*

Example

`https://example.com:443/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 "tags.read" and "push.application.<*applicationId*>" 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" : "2015-08-22T18:19:58Z",
  "description" : "Description about SampleTag",
  "lastUpdatedTime" : "2015-08-22T18:19:58Z",
  "name" : "SampleTag",
}
```

Response Properties

The response has the following properties:

createdMode
Defaults to API

createdTime
The time at which the tag was created

description
The description of the tag

lastUpdatedTime
The time at which the tag was last updated

name
An unique name of the tag in the application

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The tag with the specified tagName is not found.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Tag (PUT)

Updates tag information

Description

Updates the tag referenced by the tagName of the application referenced by the applicationId.

Method

`PUT`

Path

/apps/*applicationId*/tags/*tagName*

Example

```
https://example.com:443/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 "tags.write" and "push.application.<*applicationId*>" 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.

Produces

application/json

Payload

The details of the tag.

JSON Example

```
{
  "description" : "Description about SampleTag",
  "name" : "SampleTag",
}
```

Payload Properties

The payload has the following properties:

description
The description of the tag

name
An unique name of the tag in the application

Response

The details of the tag.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-08-22T18:19:58Z",
  "description" : "Description about SampleTag",
  "href" : "https://example.com:443/imfpush/v1/apps/testApp/tags/SampleTag",
  "lastUpdatedTime" : "2015-08-22T18:19:58Z",
  "name" : "SampleTag",
}
```

Response Properties

The response has the following properties:

createdMode
Defaults to API

createdTime
The time at which the tag was created

description
The description of the tag

href
The URL to the tag

lastUpdatedTime
The time at which the tag was last updated

name
An unique name of the tag in the application

Errors

400

`Bad Request - The request was not understood by the push server. An invalid JSON could result in thi`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The tag with the specified tagName is not found.`

405

`Unsupported Content type - The content type specified in Content-Type header is not application/json`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Tag (DELETE)

Delete the tag in the application.

Method

`DELETE`

Path

/apps/*applicationId*/tags/*tagName*

Example

```
https://example.com:443/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 "tags.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
The tag with the specified tagName is not found.
```

500

```
An internal error occurred.
```

Push Webhooks (POST)

Creates a webhook.

Description

Creates a webhook with the unique name in the application, which is referenced by the applicationId parameter. The webhook has associated with a name, url and event types. The webhook name cannot be updated after it is created

Method

```
POST
```

Path

/apps/*applicationId*/webhooks

Example

```
https://example.com:443/imfpush/v1/apps/myapp/webhooks
```

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 "webhooks.write" and "push.application.<*applicationId*>" 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.

Produces

application/json

Payload

The details of the webhook.

JSON Example

```
{
  "eventTypes" : "onDeviceRegister,onDeviceUpdate,onDeviceUnregister,onSubscribe,onUnsubscribe",
  "name" : "SampleWebhook",
  "url" : "http://samplewebhook.com",
}
```

Payload Properties

The payload has the following properties:

eventTypes
The list of event types comma separated

name
An unique name of the webhook in the application

url
The url of the webhook

Response

The details of the webhook.

JSON Example

```
{
  "eventTypes" : "onDeviceRegister,onDeviceUpdate,onDeviceUnregister,onSubscribe,onUnsubscribe",
  "name" : "SampleWebhook",
  "url" : "http://samplewebhook.com",
}
```

Response Properties

The response has the following properties:

eventTypes
The list of event types comma separated

name
An unique name of the webhook in the application

url
The description of the webhook

Errors

400

`Bad Request - The request was not understood by the push server. An invalid JSON could result in t`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

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 Webhooks (PUT)

Updates an existing webhook.

Method

PUT

Path

/apps/*applicationId*/webhooks/*webhookName*

Example

```
https://example.com:443/imfpush/v1/apps/myapp/webhooks/mywebhook
```

Path Parameters

applicationId
The name or identifier of the application

webhookName
The identifier of the webhook

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 "webhooks.write" and "push.application.<*applicationId*>" 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.

Produces

application/json

Payload

The details of the webhook.

JSON Example

```
{
  "eventTypes" : "onDeviceRegister,onDeviceUpdate,onDeviceUnregister,onSubscribe,onUnsubscribe",
  "name" : "SampleWebhook",
  "url" : "http://samplewebhook.com",
}
```

Payload Properties

The payload has the following properties:

eventTypes
The list of event types comma separated

name
An unique name of the webhook in the application

url
The url of the webhook

Response

The details of the webhook.

JSON Example

```
{
  "eventTypes" : "onDeviceRegister,onDeviceUpdate,onDeviceUnregister,onSubscribe,onUnsubscribe",
  "name" : "SampleWebhook",
  "url" : "http://samplewebhook.com",
}
```

Response Properties

The response has the following properties:

eventTypes
The list of event types comma separated

name
An unique name of the webhook in the application

url
The description of the webhook

Errors

400

`Bad Request - The request was not understood by the push server. An invalid JSON could result in t`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The webhook does not exist.`

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 Webhook (DELETE)

Delete the webhook in the application.

Method

`DELETE`

Path

/apps/*applicationId*/webhooks/*webhookName*

Example

`https://example.com:443/imfpush/v1/apps/myapp/webhooks/mywebhook`

Path Parameters

applicationId
The name or identifier of the application

webhookName
The identifier of the webhook

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 "webhooks.write" and "push.application.<*applicationId*>" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The webhook with the specified name is not found.`

500

`An internal error occurred.`

Push Health Checker (GET)

Checks the status of Push Service.

Method

`GET`

Path

/health/status

Example

`https://example.com:443/imfpush/v1/health/status`

Header Parameters

Some header parameters are optional.

`Accept-Language`
(Optional) The preferred language to use for error messages. Default:en-US

Produces

application/json

Errors

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Bulk Push Messages (POST)

Send bulk messages with different options that you can specify.

Method

`POST`

Path

/apps/*applicationId*/messages/bulk

Example

```
https://example.com:443/imfpush/v1/apps/myapp/messages/bulk
```

Path Parameters

`applicationId`
The name or identifier of the application

Header Parameters

Some header parameters are optional.

`Authorization`
The token with the scope "messages.write" and "push.application.<*applicationId*>" 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",
      },
      "notificationType" : 1,
      "settings" : {
        "apns" : {
          "badge" : 1,
          "iosActionKey" : "Ok",
          "payload" : {"custom":"data"},
          "sound" : "song.mp3",
          "type" : "SILENT",
        },
        "gcm" : {
          "delayWhileIdle" : false,
          "payload" : {"custom":"data"},
          "sound" : "song.mp3",
          "timeToLive" : 10,
        },
        "wns" : {
          "badge" : {"value":"10"},
          "cachePolicy" : false,
          "expirationTime" : 20,
          "raw" : {"payload":{"custom":"data"}},
          "tile" : {"visual":{"binding":[{"template":"TileSquareText04", "text": [{"content":"Text
          "toast" : {"launch":{"custom":"data"}, "visual":{"binding":{"template":"ToastText04","text
        },
      },
      "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

notificationType
: Integer value to indicate the channel (Push/SMS) used to send message. Allowed values are 1 (only Push), 2 (only SMS) and 3 (Push and SMS)

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.

wns
: Attributes for sending message to a 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 *wns* has the following properties:

badge

cachePolicy
: A boolean value that indicates if the notification should be cached or not.

expirationTime
: Optional. Expriry time of the notification.

raw

tile

toast

The *badge* has the following properties:

value
: Optional. A numeric or string value that indicates a prdefined glyph to be displayed.

version
: Optional. Version of the payload.

The *raw* has the following properties:

payload
: Optional. A JSON block that is transferred to the application only if the application is already open.

The *tile* has the following properties:

tag
Optional. A string value that is set as label for the notification. Used in notification cycling.

visual

The *visual* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

baseUri
Optional. Base URI to be combined with the relative URIs.

binding
For tile notifications, its a JSON array containing JSON blocks of binding attributes. For toast notification, its a JSON block of binding attributes.

branding
Optional. Indicates whether logo or app's name to be shown. Default is None.

contentId
Optional. A string value that identifies the notification content. Only applies to tile notifications.

lang
Optional. Locale of the payload.

version
Optional. Version of the payload.

The *binding* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

baseUri
Optional. Base URI to be combined with the relative URIs.

branding
Optional. Indicates whether logo or app's name to be shown. Default is None.

contentId
Optional. A string value that identifies the notification content. Only applies to tile notifications.

fallback
Optional. Template to be used as a fallback.

image
Optional. A JSON array containing JSON blocks of following image attributes.

lang
Optional. Locale of the payload.

template
Mandatory. Template type of the notification.

text
Optional. A JSON array containing JSON blocks of following text attributes.

The *image* has the following properties:

addImageQuery
Optional. A boolean value that indicates if the query string need to be appended to image URI.

alt
Optional. Image description.

src
Mandatory. Image URI.

The *text* has the following properties:

content
Mandatory. A string value that is displayed in the toast.

lang
Optional. Locale of the payload.

The *toast* has the following properties:

audio

duration
Optional. Notification will be displayed for the specified duration. Should be 'short' or 'long'.

launch
Optional. A string value that is passed to the application when it is launched by tapping or clicking the toast notification.

visual

The *audio* has the following properties:

loop
Optional. A boolean value to indicate if the sound should be repeated or not.

silent
Optional. A boolean value to indicate if the sound should be played or not.

src
Optional. A string value that specifies the notification sound type or path to local audio file.

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 W (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

`Bad Request - The request was not understood by the push server. An invalid JSON could result in thi`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`The application does not exist.`

500

`An internal error occurred.`

REST API for the MobileFirst runtime

The REST API for the MobileFirst runtime provides several services for mobile clients and confidential clients to call adapters, obtain access tokens, get Direct Update content, and more.

All the APIs in the following classes access the REST API of the runtime behind the scenes:

- WL.Client (Objective-C)
- WL.Client (Java)
- WL.Client (JavaScript)

Most of the REST API endpoints are protected by OAuth.

Testing the REST API for the MobileFirst runtime with Swagger UI

On a development server, you can test the runtime REST API with Swagger UI. MobileFirst Development Server exposes the runtime REST API at the `/doc` endpoint:

`http(s)://`*`<server_ip>`*`:`*`<server_port>`*`/`*`<context_root>`*`/doc`

REST API for MobileFirst Analytics and Logger

The MobileFirst Analytics public REST API is documented in Swagger.

To view and interact with Swagger, deploy the `analytics-service.war` file and go to the context root in your browser.

`http://`*`<hostname>`*`:`*`<port>`*`/analytics-service`

For more information about how to deploy the `analytics-service.war` file, see "MobileFirst Analytics Server installation guide" on page 11-2.

Deploying MobileFirst Server to the cloud

You can deploy MobileFirst Server to the cloud. Review the various options to run MobileFirst Server on the cloud.

You can use the IBM Mobile Foundation for Bluemix service to provision and orchestrate IBM MobileFirst Platform Foundation on the cloud. You can quickly set up a MobileFirst Server environment on Bluemix by using the Mobile Foundation service, from where you can create and run enterprise mobile apps.

For more information about how to use the IBM Mobile Foundation for Bluemix service, see Getting started with Mobile Foundation.

Deploying to the cloud

You can deploy MobileFirst applications to the cloud as Liberty for Java application on Cloud Foundry or as applications on IBM Containers. This type of deployment enables you to deploy what is developed in IBM MobileFirst Platform Foundation to a cloud platform such as IBM Bluemix.

The IBM MobileFirst Platform Foundation offerings is provided by using Liberty for Java on Cloud Foundry or by using 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 on cloud

Using V8.0.0, you can run instances of MobileFirst Server and MobileFirst Analytics in IBM Containers on IBM Bluemix. Alternatively, you can also run instances of the MobileFirst Server on Cloud Foundry as Liberty for Java application on IBM Bluemix and connect them to MobileFirst Analytics instances deployed in IBM Containers on IBM Bluemix.

Supported operating systems include Linux and Mac OS X.

V8.0.0 package overview

The V8.0.0 package contains the artifacts to create a MobileFirst Server as a Cloud Foundry Liberty for Java application or as an instance in IBM Containers, a MobileFirst Analytics container, and the components necessary for configuring and deploying them to IBM Bluemix.

- The MobileFirst Server offering contains the following product components:
  - IBM MobileFirst Platform Server
  - IBM MobileFirst Platform Operations Console

  These product components are deployed either as Liberty for Java application or are deployed in IBM Containers.
- The MobileFirst Analytics contains the following product components:
  - IBM MobileFirst Analytics server
  - IBM MobileFirst Analytics console

  These product components can be deployed only in IBM Containers.
- More components:

  - Liberty for Java runtime
  - Configuration files
  - Scripts to build and deploy

You customize your product components in the containers before they are built and deployed to the IBM Containers service on Bluemix.

See also “Package structure and contents”

Note: The Reports database is deprecated and does not work with IBM MobileFirst Platform Foundation on IBM Containers. Instead, use a MobileFirst Analytics container. Ensure that the project WAR files in your containers do not have any artifacts or configurations that are related to the deprecated Reports database.

Using the IBM MobileFirst Platform Foundation on Liberty for Java Cloud Foundry application

Go through the following main steps for using V8.0.0.
1. Customizing the product components included in the offering `.zip`.
2. Uploading and deploying the Cloud Foundry application with your customization.
3. Running the Cloud Foundry application on IBM Bluemix.

Using the IBM MobileFirst Platform Foundation Containers

Go through the following main steps for using V8.0.0.
1. Customizing the product components included in the container.
2. Building the container image format with your customization.
3. Deploying and running the built images on the IBM Containers service.

Prerequisites

The following elements are required.
- A Java Runtime. 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

  Note: This installation is optional and is required if you want to install the MobileFirst Server in IBM Containers or if you want to install MobileFirst Analytics.
- An IBM Bluemix account

Package structure and contents

The V8.0.0 `ibm-mfpf-container-8.0.0.0.zip` package unpacks to the scripts and other necessary contents that deploy to IBM Bluemix.

The V8.0.0 package provides the 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-8.0.0.0.zip folder, also referred to as *package_root* in this document. The following tables contain descriptions of the top-level folders of the package offering.

Folders

Table 9-1. Top-level folders

Folder	Description
dependencies	Contains the IBM MobileFirst Platform Foundation runtime and IBM Java JRE 8.
mfpf-analytics	Contains the artifacts required to build and deploy the MobileFirst 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.
mfpf-server-libertyapp	Contains the artifacts required to build and deploy a MobileFirst Server as Liberty for Java Cloud Foundry application.

Table 9-2. MobileFirst Analytics container and MobileFirst Server for Cloud Foundry and container folders. The subfolder names within the MobileFirst Analytics container `mfpf-server` and MobileFirst Server container `mfpf-server` folder

Folder	Description
scripts	Contains the properties files and scripts for building and deploying MobileFirst Server as a Cloud Foundry application or deploying in the IBM Containers and for deploying MobileFirst Analytics in IBM 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 build and run:
- MobileFirst Server as Liberty for Java Cloud Foundry application
- MobileFirst Server in IBM Containers
- MobileFirst Analytics in IBM 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-server-libertyapp/scripts/args

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 V8.0.0

To set up MobileFirst Server V8.0.0, you can follow the procedure described in this topic..

Before you begin

Install the required programs and verify that you have met the requirements listed in the Prerequisites section.

Procedure

1. Download the V8.0.0 package and extract the contents to a folder in your development environment.

   This folder is referred to as your installation directory or *package_root*.
2. Optional: Set the namespace for the container image registry:
   - To get the existing namespace, run the **`cf ic namespace get`** command.
   - To set the namespace, run the **`cf ic namespace set`**"*your_namespace*" command.

   For more information, see Managing images.

   Note: This step is required only if you are deploying MobileFirst Server on IBM Containers or MobileFirst Analytics on IBM Containers.
3. Customize the product components in the container as needed.

   For more information, see "Customizing MobileFirst Server containers" on page 9-12.
4. Configure security aspects as explained in Configure security.
5. Build the images.
   a. Optional: If you are using MobileFirst Analytics, build this image first and then deploy it to the IBM Containers service on IBM Bluemix.
   b. Build the MobileFirst Server container image and then deploy it to the IBM Containers service on IBM Bluemix.
   c. Build the MobileFirst Server Cloud Foundry application on Liberty and then deploy it to IBM Bluemix.
6. Deploy your customized container images or Cloud Foundry applications to Bluemix.
7. Run the Cloud Foundry application or the container.

MobileFirst Server as Liberty for Java Cloud Foundry application on IBM Bluemix

You can setup and configure the MobileFirst Server V8.0.0 as a Liberty for Java Cloud Foundry application on IBM Bluemix using the procedure described here.

The IBM MobileFirst Platform Foundation offerings is provided by using Liberty for Java on Cloud Foundry on IBM Bluemix.

You can setup and configure the MobileFirst Server V8.0.0 as a Liberty for Java Cloud Foundry application on IBM Bluemix using the procedure described here.

Customizing MobileFirst Server Liberty for Java Cloud Foundry application:

You can customize the MobileFirst Server settings before building MobileFirst Server Liberty for Java Cloud Foundry application.

The MobileFirst Server gets created from the artifacts that are provided in the V8.0.0 package. For an overview of the package contents and folder structure, see "Package structure and contents" on page 9-2.

The customizable elements for the MobileFirst Server for Liberty for Java Cloud Foundry applications are located in *package_root*/`mfpf-server-libertyapp/usr`. The following tables describe the subfolders and files to use for customization.

Table 9-3. Descriptions of the `mfpf-server-libertyapp/` sub folders

Folder	**Description**
`./usr`	Contains the customization template for the MobileFirst Server.
`./usr/config`	Contains the server configuration elements (keystore, server properties, user registry) used by MobileFirst Server.
File name	**Description**
`keystore.xml`	The configuration of the repository of security certificates used for SSL encryption. The files listed must be referenced from 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 service" on page 6-174 • "List of JNDI properties for MobileFirst runtime" on page 6-183
`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`.
`tracespec.xml`	Trace specification for Liberty.
`ltpa.xml`	Specifies the location of LTPA keys meant for single sign-on.

`./usr/env`
Contains the environment properties used for server initialization (`server.env`) and custom JVM options `jvm.options`. See Table 9-4 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/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.

Table 9-4. 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 9-4. 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_ADMIN_ROOT	mfpadmin	The context root at which the MobileFirst Server Administration Services are made available.
MFPF_CONSOLE_ROOT	mfpconsole	The context root at which the MobileFirst Operations Console is made available.
MFPF_ADMIN_GROUP	mfpadmingroup	The name of the user group assigned the predefined role *mfpadmin*.
MFPF_DEPLOYER_GROUP	mfpdeployergroup	The name of the user group assigned the predefined role *mfpdeployer*.
MFPF_MONITOR_GROUP	mfpmonitorgroup	The name of the user group assigned the predefined role *mfpmonitor*.
MFPF_OPERATOR_GROUP	mfpoperatorgroup	The name of the user group assigned the predefined role *mfpoperator*.
MFPF_SERVER_ADMIN_USER	WorklightRESTUser	The Liberty server administrator user for MobileFirst Server Administration Services.
MFPF_SERVER_ADMIN_PASSWORD	mfpadmin 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.

*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 Analytics, you must build and run the MobileFirst Analytics container before deploying and running the MobileFirst Server.

Port number limitation

You will not have the choice of changing the port number, since Cloud Foundry applications on IBM Bluemix are always routed through the Bluemix router. The default ports cannot be changed. The port 80 (HTTP) or 443 (HTTPS) are to be used.

Building and running the MobileFirst Server:

Use the scripts that are provided to build and run your customized server. You can find the scripts in the V8.0.0 package installation directory under `mfpf-server-libertyapp/scripts`.

Before you begin

- You finished customizing the server.

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-server-libertyapp`/scripts/args`

  Example command execution usage: `prepareserver.sh args/prepareserver.properties`

Procedure

Based on the database type used you can follow one of the options below.

1. Create a Bluemix database service instance.

   You can create a database service instance on Bluemix in two ways:

   - Using the Bluemix dashboard.
   - Using the Cloud Foundry command line utility.

   To create a service instance of dashDB 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. Click **USE SERVICES OR APIS**.

   d. Search for **dashDB**, from the services catalog.

   e. Select a value of **Leave unbound** for the **App** field. Enter a name for the service instance in the **Service name** field. Select an **Enterprise Transactional Plan** for the service and click **CREATE**.

      Note:

dashDB Enterprise Transactional Bluemix plans are the only dashDB plans supported. dashDB Transactional plans currently available are:

- Enterprise Transactional 2.8.500
- Enterprise Transactional 12.128.1400

The dashDB database service instance is created on Bluemix.

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.

-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
```

Note: The deployment of the dashDB Enterprise Transactional plans may not be immediate. You might be contacted by the Sales team before the deployment of the service.

2. Bring your own IBM DB2 database.

   You can choose to use your own instance of DB2 database, perform the following steps to configure DB2 database :

   a. Set up your DB2 server.
   b. Create a database on the DB2 server. You can refer to "DB2 database and user requirements" on page 6-65, for more information.
   c. Make a note of the following regarding your DB2 installation.

Parameters	Description
Host	Hostname where the DB2 is setup. This host should be accessible from the machine where the scripts are run, as well as from Bluemix where the MobileFirst Server server will be started.
Database	The database name.
Port	Port number for the database.
Username	Username for the database user. You will need to ensure that the user has correct permissions to create tables under the **Schema name** provided.
Password	Password for the database user.
Schema name	Name of the schema where you would like the scripts to create the database tables. If the schema does not already exist, the scripts will attempt to create it.

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 prepares the dashDB database service instance by creating the required tables and also configures the MobileFirst Server to use the database.

   Supported options include dashDB service (Transactional Plans) or DB2 (bring your own DB2 database).

   If your choose dashDB, then Bluemix database service instance name **Service name**, created in Step 1, is supplied as an argument to this script.

   If you select to use IBM DB2, then make a note of the database details with you, to supply as input parameters to the script.
   You can optionally specify a database schema name. The default schema name is `MFPDATA`.

   prepareserver.sh
   This script builds the server application with the `mfp-server-libertyapp` customizations and pushes the application to IBM Bluemix.

   Provide a value for the mandatory argument, **-n|--name** [*APP_NAME*]

   startserver.sh
   The script starts the Cloud Foundry application built and deployed in the previous step.

Script overview and usage:

Use the scripts for configuring, building, and deploying a MobileFirst Server as a Liberty for Java Cloud Foundry application.

The scripts are located in the `package_root/mfpf-server-libertyapp/scripts` folder.

Environment initialization script to build and run MobileFirst Server | initenv.sh:

This script file creates the environment for building and running a MobileFirst Server and performs the tasks necessary for logging into IBM Bluemix. Run this script before running any other scripts for building and deploying IBM MobileFirst Platform Foundation.

Table 9-5. 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 9-6. Optional command-line arguments

Command-line argument	Description
[**-a**|**--api**] *BLUEMIX_API_URL*	Bluemix API endpoint (Defaults to `https://api.ng.bluemix.net`)

Script to configure databases | prepareserverdbs.sh script:

This script configures MobileFirst Server databases (administration, configuration, runtime and push). You must run this script before creating the Liberty for Java Cloud Foundry application.

Table 9-7. Mandatory command-line arguments

Command-line argument	Description
[**-t** |**--type**] *DB_TYPE*	Database type used (DB2 | dashDB). Bluemix dashDB service (with Bluemix service plan of **Enterprise Transactional**)

Usage example:

```
prepareserverdbs.sh
 --type dashDB --admindb MFPDashDBService
```

Table 9-8. Optional command-line arguments

Command-line argument	Description
[**-h** |**--host**] *DB2_HOST*	Hostname or IP address where the database is set up. Required if *DB_TYPE* is DB2.

Table 9-8. Optional command-line arguments (continued)

Command-line argument	Description
[**-d** |**--database**] *DB2_DATABASE*	Name of the database. Required if *DB_TYPE* is DB2.
[**-r** |**--port**] *DB2_PORT*	Port number for DB2. Required if *DB_TYPE* is DB2.
[**-u** |**--username**] *DB2_USERNAME*	Username for the DB2 user. Required if *DB_TYPE* is DB2.
[**-pw** |**--password**] *DB2_PASSWORD*	Password for the DB2 user. Required if *DB_TYPE* is DB2.
[**-adl** |**--admindb**] *ADMIN_DB_SRV_NAME*	Bluemix dashDB service (with Bluemix service plan of **Enterprise Transactional**)
[**-as** |**--adminschema**] *ADMIN_SCHEMA_NAME*	Database schema name for administration service. **Note:** Defaults to MFPDATA
[**-rd** |**--runtimedb**] *RUNTIME_DB_SRV_NAME*	Bluemix database service instance name for storing runtime data. **Note:** Defaults to the same service as given for admin data.
[**-p** |**--push**] *ENABLE_PUSH*	Enable configuring database for push service. **Note:** Accepted values are Y (default) or N.
[**-pd** |**--pushdb**] *PUSH_DB_SRV_NAME*	Bluemix database service instance name for storing push data. **Note:** Defaults to the same service as given for runtime data.
[**-ps** |**--pushschema**] *PUSH_SCHEMA_NAME*	Database schema name for push service. **Note:** Defaults to the runtime schema name.

Script to create MobileFirst Server as a Cloud Foundry app | prepareserver.sh:

This script creates the MobileFirst Server as a Liberty for Java Cloud Foundry application and pushes it to IBM Bluemix. Ensure that you run the `prepareserverdbs.sh` before running this script.

Table 9-9. Mandatory command line arguments

Command line argument	Description
[**-n**|**--name**] *APP_NAME*	Name to be used for the customized MobileFirst Server Cloud Foundry application.

Usage example:

```
prepareserver.sh --name mobilefirst80app
```

Script to run MobileFirst Server as a Cloud Foundry app | startserver.sh:

This script runs the MobileFirst Server as a Liberty for Java Cloud Foundry application on IBM Bluemix. Ensure that you have run the `prepareserver.sh` script to upload the application to IBM Bluemix before running this script.

Table 9-10. Mandatory command line arguments

Command line argument	Description
[**-n**|**--name**] **APP_NAME**	Name of the MobileFirst Server application

Usage example:

```
startserver.sh
 --name mobilefirst80app
```

Table 9-11. Optional command line arguments

Command line argument	Description
[**-h**|**--host**] *APP_HOST*	The hostname for the application route to be created. The application name (*APP_NAME*) is taken as the default value for the hostname.
[**-d**|**--domain**] *DOMAIN_NAME*	Domain name used in the application route. `mybluemix.net` is taken as the default value.
[**-m**|**--memory**] *SERVER_MEM*	Assign a memory size limit to the container in megabytes (MB). Accepted values are 1024 MB (default) and 2048 MB.
[**-i**|**--instances**] *INSTANCES*	The desired number of nodes (instances) of the application that you want to create. The default is a 2 node application.

MobileFirst Server container

This section contains the information you need to configure and run a MobileFirst Server container.

Customizing MobileFirst Server containers:

You can customize the MobileFirst Server settings before building MobileFirst Server container images.

The container gets created from the artifacts that are provided in the V8.0.0 package. For an overview of the package contents and folder structure, see "Package structure and contents" on page 9-2.

The customizable elements for the MobileFirst Server container are located in *package_root*/`mfpf-server/usr`. The following tables describe the subfolders and files to use for customization.

Table 9-12. 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 9-12. 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 service" on page 6-174 "List of JNDI properties for MobileFirst runtime" on page 6-183 • `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 9-13 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/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.

Table 9-13. 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.
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 `Farm` value is automatically set when the container is run as a container group.
MFPF_ADMIN_ROOT	mfpadmin	The context root at which the MobileFirst Server Administration Services are made available.
MFPF_CONSOLE_ROOT	mfpconsole	The context root at which the MobileFirst Operations Console is made available.
MFPF_ADMIN_GROUP	mfpadmingroup	The name of the user group assigned the predefined role *mfpadmin*.

Table 9-13. Supported server environment properties (`server.env`) (continued)

Property	Default Value	Description
MFPF_DEPLOYER_GROUP	mfpdeployergroup	The name of the user group assigned the predefined role *mfpdeployer*.
MFPF_MONITOR_GROUP	mfpmonitorgroup	The name of the user group assigned the predefined role *mfpmonitor*.
MFPF_OPERATOR_GROUP	mfpoperatorgroup	The name of the user group assigned the predefined role *mfpoperator*.
MFPF_SERVER_ADMIN_USER	WorklightRESTUser	The Liberty server administrator user for MobileFirst Server Administration Services.
MFPF_SERVER_ADMIN_PASSWORD	mfpadmin 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.

*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 Analytics, you must build and run the MobileFirst 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 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 V8.0.0 package installation directory under `mfpf-server/scripts`.

Before you begin

- You 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-server-libertyapp``/scripts/args`

  `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 Bluemix database service instance.

   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 the database service instance.

   Using the Bluemix dashboard:

To create a service instance of dashDB 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. Click **USE SERVICES OR APIS**.
d. Search for **dashDB**, from the services catalog.
e. Select a value of **Leave unbound** for the **App** field. Enter a name for the service instance in the **Service name** field. Select an **Enterprise Transactional Plan** for the service and click **CREATE**.

   Note:

   dashDB Enterprise Transactional Bluemix plans are the only dashDB plans supported. dashDB Transactional plans currently available are:
   - Enterprise Transactional 2.8.500
   - Enterprise Transactional 12.128.1400

   The dashDB database service instance is created on Bluemix.

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.

   `-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
   ```

Note: The deployment of the dashDB Enterprise Transactional plans may not be immediate. You might be contacted by the Sales team before the deployment of the service.

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 prepares the dashDB database service instance by creating the required tables and also configures the MobileFirst Server to use the database.

  Your Bluemix database service instance name **`Service name`**, created in Step 2, is supplied as an argument to this script.

  You can optionally specify a database schema name. The default schema name is MFPDATA.

`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/mfpserver*

`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=`*`mfp.analytics.url:http://127.0.0.1/analytics-service/rest`*`,`*`mfp.analytics.console.url:http://127.0.0.1/analytics/console`* 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.

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.

Environment initialization script to build and run MobileFirst Server | initenv.sh:

This script file creates the environment for building and running a MobileFirst Server and performs the tasks necessary for logging into IBM Bluemix. Run this script before running any other scripts for building and deploying IBM MobileFirst Platform Foundation.

Table 9-14. 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 9-15. Optional command-line arguments

Command-line argument	Description
[**-a**|**--api**] *BLUEMIX_API_URL*	Bluemix API endpoint (Defaults to `https://api.ng.bluemix.net`)

Script to configure databases | prepareserverdbs.sh script:

This script configures MobileFirst Server databases (administration, configuration, runtime and push). You must run this script before creating the container image.

Table 9-16. Mandatory command-line arguments

Command-line argument	Description
[**-adl** |**--admindb**] *ADMIN_DB_SRV_NAME*	Bluemix dashDB service (with Bluemix service plan of **Enterprise Transactional**)

Usage example:

```
prepareserverdbs.sh
 --admindb MFPDashDBService
```

Table 9-17. Optional command-line arguments

Command-line argument	Description
[**-as** | **--adminschema**] *ADMIN_SCHEMA_NAME*	Database schema name for administration service. **Note:** Defaults to MFPDATA
[**-rd** | **--runtimedb**] *RUNTIME_DB_SRV_NAME*	Bluemix database service instance name for storing runtime data. **Note:** Defaults to the same service as given for admin data.
[**-p** | **--push**] *ENABLE_PUSH*	Enable configuring database for push service. **Note:** Accepted values are Y (default) or N.
[**-pd** | **--pushdb**] *PUSH_DB_SRV_NAME*	Bluemix database service instance name for storing push data. **Note:** Defaults to the same service as given for runtime data.
[**-ps** | **--pushschema**] *PUSH_SCHEMA_NAME*	Database schema name for push service. **Note:** Defaults to the runtime schema name.

Script to create MobileFirst Server image | prepareserver.sh:

This script creates the MobileFirst Server image and pushes it to IBM Containers on Bluemix. Ensure that you run the `prepareserverdbs.sh` before running this script.

Table 9-18. 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
```

Script to run MobileFirst Server in container | startserver.sh:

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 9-19. 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 9-20. Optional command line arguments

Command line argument	Description
[**-si**|**--services**] *SERVICE_INSTANCES*	Comma-separated Bluemix service instances that you want to bind 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: *mfp.analytics.url:http://127.0.0.1/analytics-service/rest,mfp.analytics.console.url:http://127.0.0.1/analytics/console* Note: If you specify properties using this script, ensure that

Script to run MobileFirst Server in container group | startservergroup.sh:

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 9-21. 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 9-21. 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 9-22. 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).
[**-si**|**--services**] *SERVICES*	Comma-separated Bluemix service instance names that you want to bind to the container.
[**-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: *mfp.analytics.url:http://127.0.0.1/analytics-service/rest mfp.analytics.console.url:http://127.0.0.1/analytics/console* **Note:** If you specify properties using this script, ensure that the same properties have not been set in the configuration files in the `usr/config` folder.
[**-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.

MobileFirst Analytics containers

This section contains the information you need to configure and run MobileFirst Analytics containers.

Customizing MobileFirst Analytics containers:

Use the scripts provided in the V8.0.0 package to customize container images.

The folder where you extracted the contents of the V8.0.0 package is referred to in this document as the *installation directory*. The customizable elements for the MobileFirst Analytics container are located in `package_root/mfpf-analytics/usr`. See the following tables for details.

Table 9-23. Descriptions of the `mfpf-analytics/` sub directories

Folders and Files	Description
`./usr`	The root folder that contains the customization template for the MobileFirst 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 Analytics container.
`./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 Analytics can be provided here. See the list of supported properties at "Configuration properties" on page 11-15.
`./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 9-24. 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 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 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/prepareanalytics.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 Analytics container image.

The scripts are located in the *`package_root`*`/mfpf-analytics/scripts` folder.

Environment initialization script to build and run MobileFirst Analytics in a container | initenv.sh:

This script file creates the environment for building and running a MobileFirst 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 Analytics container images.

Table 9-25. 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 9-26. Optional command-line arguments

Command-line argument	**Description**
[**-a**|**--api**] *BLUEMIX_API_URL*	Bluemix API endpoint (Defaults to `https://api.ng.bluemix.net`)

Script to create MobileFirst Analytics image | prepareanalytics.sh:

This script creates the MobileFirst Analytics image and pushes it to IBM Containers on Bluemix. Ensure that you have run the `initenv.sh` before running this script.

Table 9-27. 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/mfpfanalytics7
```

Script to run MobileFirst Analytics in a container | startanalytics.sh:

This script runs the MobileFirst 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 9-28. 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 **ice ip request** command.)

Usage example:

```
startanalytics.sh
 --tag image_tag_name
 --name container_name
 --ip container_ip_address
```

Table 9-29. 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*

Table 9-29. Optional command-line arguments (continued)

Command-line argument	Description
[**-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).
[**-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 Analytics properties as comma-separated key:value pairs. Note: If you specify properties using this script, ensure that

Script to run MobileFirst Analytics in a container group | startanalyticsgroup.sh:

This script runs a MobileFirst 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 9-30. 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 9-31. Optional command-line arguments

Command-line argument	Description
[**-gm|--min**] **ANALYTICS_CONTAINER_GROUP_MIN**	The minimum number of container instances. The default value is

Table 9-31. Optional command-line arguments (continued)

Command-line argument	Description
[**-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).
[**-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: *mfp.analytics.url:http://127.0.0.1/analytics-service/rest/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).
[**-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"

Note: Container Group for MobileFirst Analytics requires the analytics data to be shared across instances. The **startanalyticsgroup.sh** command automatically creates volumes with the names provided in the command.

Learn more about creating scalable container groups.

Securing containers

Security configuration for IBM MobileFirst Platform Foundation on IBM Containers:

Your IBM MobileFirst Platform Foundation 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 9-28 for instructions.

Application-authenticity validation

To keep unauthorized mobile applications from accessing the MobileFirst Server, enable the application-authenticity security check. 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 Analytics container, you must include the private IP address of the MobileFirst Analytics container in the `mfpfProperties.xml` file.

Before you begin

To complete this task, you need the private IP of the MobileFirst 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 Analytics, you must configure, build, and run the MobileFirst 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 `mfp.analytics.url` property to the private IP address of the MobileFirst Analytics container. Example:
   ```
   <jndiEntry jndiName="mfp.analytics.url" value="http://AnalyticsContainerPrivateIP:9080/analyti
   ```
   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 Analytics console can be accessed on the network, set the `mfp.analytics.console.url` property to the public IP address of the MobileFirst Analytics container. Example:
   ```
   <jndiEntry jndiName="mfp.analytics.console.url" value="http://AnalyticsContainerPublicIP:9080/
   ```

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 9-15 or “Building and running the MobileFirst Analytics container” on page 9-23. 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 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 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:
- Setup 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 9-32. 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 a secure gateway to connect to a local LDAP repository and the container. (You need an existing app on Bluemix for this step.)

Procedure

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.

1. Log on to Bluemix and navigate to **Catalog**, **Category** > **Integration**, and then click **Secure Gateway**.
2. Under Add Service, select an app and then click **CREATE**. Now the service is bound to your app.

3. Go to the Bluemix dashboard for the app, click on the **Secure Gateway** service instance, and then click **ADD GATEWAY**.
4. Name the gateway and click **ADD DESTINATIONS** and enter the name, IP address, and port for your local LDAP server.
5. Follow the prompts to complete the connection. To see the destination initialized, navigate to the Destination screen of the LDAP gateway service.
6. 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.
7. 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 from Bluemix

If you ran the **`prepareserverdbs.sh`** script during the configuration of your V8.0.0 image, the configurations and database tables required for MobileFirst Server are created. This script also creates the database schema.

About this task

To remove the database service configuration from Bluemix, perform the following procedure using Bluemix dashboard.

Procedure

1. From the Bluemix dashboard, select the dashDB service you have used while configuring the IBM MobileFirst Platform Foundation on IBM Containers.

   Tip: Choose the dashDB service name that you had provided as parameter while running the **`prepareserverdbs.sh`** script.
2. **Launch** the dashDB console to work with the schemas and database objects of the selected dashDB service instance.
3. Select the schemas related to IBM MobileFirst Platform Foundation configuration on IBM Containers.

Tip: The schema names are ones that you have provided as parameters while running the **prepareserverdbs.sh** script.

4. Delete each of the schema after carefully inspecting the schema names and the objects under them. The database configurations are removed from Bluemix.

Log and trace collection

IBM Containers for Bluemix provides some built-in logging and monitoring capabilites around container CPU, memory, and networking. You can optionally change the log levels for your MobileFirst containers.

The option to create log files for the MobileFirst Server and MobileFirst Analytics containers is enabled by default (using level **=info*). You can change the log levels by either adding a code override manually or by injecting code using a given script file.

Both container logs and server or runtime logs can be viewed from a Bluemix logmet console by means of the Kibana visualization tool. Monitoring can be done from a Bluemix logmet console by means of Grafana, an open source metrics dashboard and graph editor.

When your IBM MobileFirst Platform Foundation container is created with a Secure Shell (SSH) key and bound to a public IP address, a suitable private key can be used to securely view the logs for the container instance.

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 V8.0.0 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`*

Accessing log files:

Logs are created for each container instance. You can access log files using the IBM Container Cloud Service REST API, by using **cf ic** commands, or by using the Bluemix logmet console.

IBM Container Cloud Service REST API

For any container instance, the docker.log and /var/log/rsyslog/syslog can be viewed using the Bluemix logmet service (https://logmet.ng.bluemix.net/kibana/). The log activities can be seen using the Kibana dashboard of the same.

IBM Containers CLI commands (`cf ic exec`) can be used to gain access to running container instances. Alternatively, you can obtain container log files through Secure Shell (SSH).

Enabling SSH

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 `startserver.sh` or `startanalytics.sh` scripts. You can optionally customize the related script `.properties` files to include the key content.

The container logs endpoint gets `stdout` logs with the given ID of the container instance.

Example: `GET /containers/{*container_id*}/logs`

```
  X-Auth-Token - Bluemix JWT token (not prepended with 'bearer')

   X-Auth-Project-Id - Space GUID.
```

Accessing containers from the command line:

You can access running MobileFirst Server and MobileFirst 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/mfp
   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/mfp/messages.log" > /tmp/local_messages.log
   ```

Accessing containers using SSH:

You can get the syslogs and Liberty logs by using Secure Shell (SSH) to access your MobileFirst Server and MobileFirst Analytics containers.

Before you begin

- SSH must be enabled for the container.

  Tip: The SSH public key must be copied to the `mfp-server\server\ssh` folder before you run the `startserver.sh` script.
- Volume has been enabled so that the log files are persisted.
- You need the public IP address of the container.

If you are running a container group, you can bind a public IP address to each instance and view the logs securely using SSH. To enable SSH, make sure to copy the SSH public key to the `mfp-server\server\ssh` folder before you run the `startservergroup.sh` script.

Procedure

- Make an SSH request to the container. Example: *mylocal-workstation#* `ssh -i` *~/ssh_key_directory/id_rsa root@public_ip*
- Archive the log file location. Example:

  ```
  container_instance@root# cd /opt/ibm/wlp/usr/servers/mfp
  container_instance@root# tar czf logs_archived.tar.gz logs/
  ```
- 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/mfp/logs_archived.tar.gz` */local_workstation_dir/target_location/*

Container log files:

This topic contains information about V8.0.0 log file locations and persistence.

Log files are generated for MobileFirst Server and Liberty Profile runtime activities for each container instance and can be found in the following locations:
- `/opt/ibm/wlp/usr/servers/mfp/logs/messages.log`
- `/opt/ibm/wlp/usr/servers/mfp/logs/console.log`
- `/opt/ibm/wlp/usr/servers/mfp/logs/trace.log`
- `/opt/ibm/wlp/usr/servers/mfp/logs/ffdc/*`

You can log in to the container by following the steps in Accessing log files and access the log files.

To persist log files, even after a container no longer exists, enable volume. (Volume is not enabled by default.) Having volume enabled can also allow you to view the logs from Bluemix using the logmet interface (such as https://logmet.ng.bluemix.net/kibana).

Enabling volume

Volume allows for containers to persist log files. The volume for MobileFirst Server and MobileFirst Analytics container logs is not enabled by default.

You can enable volume when running the `start*.sh` scripts by setting ENABLE_VOLUME [**-v | --volume**] to `Y`. This is also configurable in the `args/startserver.properties` and `args/startanalytics.properties` files for interactive execution of the scripts.

The persisted log files are saved in the `/var/log/rsyslog` and `/opt/ibm/wlp/usr/servers/mfp/logs` folders in the container.

The logs can be accessed by issuing an SSH request to the container.

See related information for more details on usage of **startserver.sh**, **startservergroup.sh**, **startanalytics.sh** and **startanalyticsgroup.sh**.

Related concepts:

"MobileFirst Analytics containers" on page 9-21
This section contains the information you need to configure and run MobileFirst Analytics containers.

Related reference:

"Script overview and usage" on page 9-18
Use the scripts for configuring, building, and deploying a MobileFirst Server container image.

Troubleshooting tips

Here are tips for resolving common problems that might occur.

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 mfpfsqldb.xml file:

An error occurs at the end of running the prepareserverdbs.sh script: `Error : unable to create mfpfsqldb.xml`

How to resolve

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 registry.

Explanation

The prepareserver.sh script pushes the entire IBM MobileFirst Platform Foundation 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.

Administration DB, Live Update and Push Service show up as inactive:

Administration DB, Live Update and Push Service show up as inactive or no runtimes are 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 interim fixes

Interim fixes for IBM MobileFirst Platform Foundation V8.0.0 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 V8.0.0 products
- IBM Passport Advantage
- IBM Fix Central

Resolving problems with IBM MobileFirst Platform Foundation deployed in IBM Containers or as Liberty for Java Cloud Foundry application on IBM Bluemix

When you are unable to resolve a problem encountered while working with IBM MobileFirst Platform Foundation 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 V8.0.0 or later) and any interim fixes that were applied.
- The container size selected. For example, *Medium 2GB*.
- The Bluemix dashDB database plan type. For example, *EnterpriseTransactional2.8.50*.
- The container ID
- The public IP address (if assigned)
- Versions of docker and cloud foundry

  `cf -v`

  `docker version`
- 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 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/mfp/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*.

Deploying MobileFirst Server on IBM PureApplication System

IBM MobileFirst Platform Foundation provides the capability to deploy and manage IBM MobileFirst Platform Server and MobileFirst applications on IBM PureApplication System and IBM PureApplication Service on SoftLayer.

IBM MobileFirst Platform Foundation in combination with IBM PureApplication System and IBM PureApplication Service on SoftLayer provides a simple and intuitive environment for developers and administrators, to develop mobile applications, test them, and deploy them to the cloud. This version of IBM MobileFirst Platform Foundation System Pattern provides MobileFirst runtime and artifacts support for the PureApplication Virtual System Pattern technologies that are included in the most recent versions of IBM PureApplication System and IBM PureApplication Service on SoftLayer. Classic Virtual System Pattern was supported in earlier versions of IBM PureApplication System.

Key benefits

IBM MobileFirst Platform Foundation System Pattern provides the following benefits:

- Predefined templates enable you to build patterns in a simple way for the most typical MobileFirst Server deployment topologies.

  Examples of the topologies are:
  - IBM WebSphere Application Server Liberty profile single node
  - IBM WebSphere Application Server Liberty profile multiple nodes
  - IBM WebSphere Application Server full profile single node
  - IBM WebSphere Application Server full profile multiple nodes
  - Clusters of WebSphere Application Server Network Deployment servers

  In V8.0.0 MobileFirst Application Center, deployment topologies such as:
  - IBM WebSphere Application Server Liberty profile single node
  - IBM WebSphere Application Server full profile single node
- Script packages act as building blocks to compose extended deployment topologies such as automating the inclusion of an analytics server in a pattern and flexible DB VM deployment options. WebSphere Application Server and DB2 script packages are available through the inclusion of WebSphere Application Server and DB2 pattern types.
- Optional JNDI properties in the runtime deployment script package allow fine-grained tuning for the deployment topology. In addition, deployment topologies that are built with IBM WebSphere Application Server full profile

now support accessing the WebSphere Application Server Administration Console, which gives you full control over the configuration of the application server.

Important restrictions

Depending on the pattern template you use, do not change some of the component attributes. If you change any of these component attributes, the deployment of patterns that are based on these templates fails.

MobileFirst Platform (Application Center Liberty single node)
Do not change the values for the following attributes in the **Liberty profile server**:
- **WebSphere product Installation directory**
- **Configuration data location**
- **Liberty profile server name**
- Under **Install an IBM Java SDK**, select only **Java SDK V7.0** or **Java SDK V7.1**
- Select the **Install additional features** and clear the selection of **IBM WebSphere eXtreme Scale**.

MobileFirst Platform (Application Center WebSphere Application Server single node)
Do not change the values for the following attributes in the **Liberty profile server**:
- **WebSphere product Installation directory**
- **Configuration data location**
- **Cell name**
- **Node name**
- **Profile name**
- Under **Install an IBM Java SDK**, select only **Java SDK V7.0** or **Java SDK V7.1**
- Select the **Install additional features** and clear the selection of **IBM WebSphere eXtreme Scale**.

MobileFirst Platform (Liberty single node)
Do not change the values for the following attributes in the **Liberty profile server**:
- **WebSphere product Installation directory**
- **Configuration data location**
- **Liberty profile server name**
- Under **Install an IBM Java SDK**, select only **Java SDK V7.0** or **Java SDK V7.1**
- Select the **Install additional features** and clear the selection of **IBM WebSphere eXtreme Scale**.

MobileFirst Platform (Liberty server farm)
Do not change the values for the following attributes in the **Liberty profile server**:
- **WebSphere product Installation directory**
- **Configuration data location**
- **Liberty profile server name**
- Under **Install an IBM Java SDK**, select only **Java SDK V7.0** or **Java SDK V7.1**

- Select the **Install additional features** and clear the selection of **IBM WebSphere eXtreme Scale**.

MobileFirst Platform (WebSphere Application Server single node) template
In the **Standalone server** component of the **MobileFirst Platform Server** node, do not unlock or change the values for any of the following attributes:

- **Cell name**
- **Node name**
- **Profile name**

If you change any of these attributes, your pattern deployment fails.

MobileFirst Platform (WebSphere Application Server server farm) template
In the **Standalone server** component of the **MobileFirst Platform Server** node, do not unlock or change the values for any of the following attributes:

- **Cell name**
- **Node name**
- **Profile name**

If you change any of these attributes, your pattern deployment fails.

MobileFirst Platform (WebSphere Application Server Network Deployment) template
In the **Deployment manager** component of the **DmgrNode** node or the **Custom nodes** component of the **CustomNode** node, do not unlock or change the values for any of the following attributes:

- **Cell name**
- **Node name**
- **Profile name**

If you change any of these attributes, your pattern deployment fails.

Limitations

The following limitations apply:

- Dynamic scaling for WebSphere Application Server Liberty profile server farms and WebSphere Application Server full profile server farms is not supported. The number of server farm nodes can be specified in the pattern by setting the scaling policy but cannot be changed during run time.
- The IBM MobileFirst Platform Foundation System Pattern Extension for MobileFirst Studio and Ant command-line interface that is supported in versions earlier than V7.0 are not available in this version of IBM MobileFirst Platform Foundation System Pattern.
- IBM MobileFirst Platform Foundation System Pattern depends on WebSphere Application Server Patterns, which has its own restrictions. For more information, see Restrictions for WebSphere Application Server Patterns.
- Due to restrictions in the uninstallation of Virtual System Patterns, you must delete the script packages manually after you delete the pattern type. In IBM PureApplication System, go to **Catalog** > **Script Packages** to delete the script packages that are listed in the “Components” on page 9-47 section.

- The **MobileFirst Platform (WebSphere Application Server Network Deployment)** pattern template does not support token licensing. If you want to use this pattern, you must use perpetual licensing. All other patterns support token licensing.

Composition

IBM MobileFirst Platform Foundation System Pattern is composed of the following patterns:

- IBM WebSphere Application Server Network Deployment Patterns 2.2.0.0.
- [PureApplication Service] WebSphere 8558 for Mobile IM repository to allow the WebSphere Application Server Network Deployment Patterns to work. Contact the administrator for IBM PureApplication System to confirm that the WebSphere 8558 IM repository is installed.
- IBM DB2 with BLU Acceleration® Pattern 1.2.4.0.
- IBM MobileFirst Platform Foundation System Pattern.

Components

In addition to all components provided by IBM WebSphere Application Server Pattern and IBM DB2 with BLU Acceleration Pattern, IBM MobileFirst Platform Foundation System Pattern provides the following Script Packages:

- MFP Administration DB
- MFP Runtime DB
- MFP Server Prerequisite
- MFP Server Administration
- MFP Server Runtime Deployment
- MFP Server Application Adapter Deployment
- MFP IHS Configuration
- MFP Analytics
- "MFP Open Firewall Ports for WAS" on page 9-113
- "MFP WAS SDK Level" on page 9-114
- "MFP Server Application Center" on page 9-120

Compatibility between pattern types and artifacts created with different product versions

If you use MobileFirst Studio V6.3.0 or earlier to develop your applications, you can upload the associated runtime, application, and adapter artifacts into patterns associated with IBM MobileFirst Platform Foundation V7.0.0 and later.

Pattern types that are associated with IBM MobileFirst Platform Foundation V6.3.0 or earlier are not compatible with runtime, application, and adapter artifacts created by using MobileFirst Studio V7.0.0 and later.

For versions V6.0.0 and earlier, only the same versions of server, `.war` file, application (`.wlapp` file), and adapters are compatible.

Installing IBM MobileFirst Platform Foundation System Pattern

You use the PureApplication System Workload Console to install IBM MobileFirst Platform Foundation System Pattern.

Before you begin

You can find the `vsys.mobilefirst-8.0.0.0.tgz` file in the `mobilefirst_patterns_8.0.0.zip` file. Make sure you extract it before you start this procedure.

Procedure

1. Log in to IBM PureApplication System with an account that has permission to create new pattern types.
2. Go to **Catalog** > **Pattern Types**.
3. Upload the IBM MobileFirst Platform Foundation System Pattern `.tgz` file:
   a. On the toolbar, click **+**. The "Install a pattern type" window opens.
   b. On the Local tab, click **Browse**, select the IBM MobileFirst Platform Foundation System Pattern `.tgz` file, and then wait for the upload process to complete. The pattern type is displayed in the list and is marked as not enabled.
4. In the list of pattern types, click the uploaded pattern type. Details of the pattern type are displayed.
5. In the License Agreement row, click **License**. The License window is displayed stating the terms of the license agreement.
6. To accept the license, click **Accept**. Details of the pattern type now show that the license is accepted.
7. In the Status row, click **Enable**. The pattern type is now listed as being enabled.
8. Mandatory for PureApplication Service: After the pattern type is enabled successfully, go to **Catalog** > **Script Packages** and select script packages with names similar to "MFP ***". On the details page to the right, accept the license in the **License agreement** field. Repeat for all eleven script packages listed in the "Components" on page 9-47 section.

Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern

If you use token licensing to license IBM MobileFirst Platform Foundation, you must install IBM Rational License Key Server and configure with your licenses before you deploy the MobileFirst Platform Pattern.

Important: The **MobileFirst Platform (WAS ND)** pattern template does not support token licensing. You must be using perpetual licensing when you deploy patterns based on the MobileFirst Platform (WAS ND) pattern template. All other pattern templates support token licensing.

Your IBM Rational License Key Server must be external to your PureApplication System. MobileFirst Platform Pattern do not support the PureApplication System shared service for IBM Rational License Key Server.

In addition, you must know the following information about your Rational License Key Server to add the license key server information to your pattern attributes:

- Fully qualified host name or IP address of your Rational License Key Server
- License manager daemon (**`lmgrd`**) port
- Vendor daemon (**`ibmratl`**) port

If you have a firewall between your Rational License Key Server and your PureApplication System, ensure that both daemon ports are open in your firewall.

The deployment of MobileFirst Platform Pattern fails if the license key server cannot be contacted or if insufficient license tokens are available.

For details about installing and configuring Rational License Key Server, see IBM Support - Rational licensing start page.

Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server

You use a predefined template to deploy MobileFirst Server on a single-node WebSphere Application Server Liberty profile server.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

Token licensing requirements: If you use token licensing to license IBM MobileFirst Platform Foundation, review the requirements outlined in “Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern” on page 9-48 before you continue. The deployment of this pattern fails if the license key server cannot be contacted or if insufficient license tokens are available.

About this task

Note:

Some parameters of script packages in the template have been configured with the recommended values and are not mentioned in this section. For fine-tuning purposes, see more information about all the parameters of script packages in “Script packages for MobileFirst Server” on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see “MobileFirst Platform (Liberty single node) template” on page 9-98.

Procedure

1. Create a pattern from the predefined template:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (Liberty single node)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
   c. In the **Name** field, provide a name for the pattern.
   d. In the **Version** field, specify the version number of the pattern.
   e. Click **Start Building**.
2. Mandatory for AIX: In IBM PureApplication System running on Power®, the MobileFirst Platform DB node needs to use the AIX-specific add-on component “Default AIX add disk” to replace the “Default add disk” component in the template to support the `jfs2` file system:
   a. In the Pattern Builder, select the **MobileFirst Platform DB** node.

b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MobileFirst Platform DB** node).

c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.

d. Select the **Default AIX add disk** component and specify the following attributes:

DISK_SIZE_GB
Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

FILESYSTEM_TYPE
Supported file system in AIX. Default value: `jfs2`.

MOUNT_POINT
Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

VOLUME_GROUP
Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.

e. In the MobileFirst Platform DB node, select the **Default add disk** component, and then click the bin icon to delete it.

f. Save the pattern.

3. Optional: Configure MobileFirst Server administration. You can skip this step if you want to specify the user credential with MobileFirst Server administration privilege later during the pattern deployment configuration phase in step 9. To specify it now, complete these steps:

Note: If you want to configure administration security with an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

a. In the MobileFirst Platform Server node, click the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.

b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.

c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.

d. If you use token licensing to license IBM MobileFirst Platform Foundation, complete the following fields. If you do not use token licensing, leave these fields blank.

ACTIVATE_TOKEN_LICENSE
Select this field to license your pattern with token licensing.

LICENSE_SERVER_HOSTNAME
Enter the fully qualified host name or IP address of your Rational License Key Server.

LMGRD_PORT
Enter the port number that the license manager daemon (**`lmrgd`**) listens for connections on. The default license manager daemon port is 27000.

`IBMRATL_PORT`
Enter the port number that the vendor daemon (**`ibmratl`**) listens for connections on. The default vendor daemon port is typically 27001.

A default administration account for MobileFirst Server is created during pattern deployment.

4. Optional: Configure MobileFirst Server runtime deployment. You can skip this step if you want to specify the context root name for the runtime later during the pattern deployment configuration phase in step 9. To specify the context root name now, complete these steps:
   a. In the MobileFirst Platform Server node, click the **MFP Server Runtime Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **runtime_contextRoot** field, click the **Delete** button to clear the pattern level parameter setting.
   c. In the **runtime_contextRoot** field, specify the runtime context root name. Note that the context root name must start with a forward slash, `/`; for example, `/HelloWorld`.
5. Upload application and adapter artifacts:

   Important: When specifying the Target path for applications and adapters, make sure all the applications and adapters are placed in the same directory. For example, if one target path is `/opt/tmp/deploy/HelloWorld-common.json`, all the other target paths should be `/opt/tmp/deploy/*`.
   a. In the MobileFirst Platform Server node, click the **MFP Server Application** or **MFP Server Adapter** component. The properties of the selected component are displayed next to the canvas.
   b. In the **Additional file** field, click the **Browse** button to locate and upload the application or adapter artifact.
   c. In the **Target path** field, specify the full path for storing the artifact including its file name; for example, `/opt/tmp/deploy/HelloWorld-common.json`.
   d. If no application or adapter is to be deployed in the pattern, remove the relevant component by clicking the **X** button inside it. To get an empty MobileFirst Operations Console deployed without any app or adapter installed, remove the MFP Server Application Adapter Deployment component by clicking the **X** button inside it.
6. Optional: Add more application or adapter artifacts for deployment:
   a. From the **Assets** toolbar, expand **Software Components**, and then drag and drop an **Additional file** component onto the MobileFirst Paltform Server node in the canvas. Rename it `MobileFirst App_`*`X`* or `MobileFirst Adatper_`*`X`* (where *X* stands for a unique number for differentiation).
   b. Hover the cursor over the newly added App or Adapter component, and then click the **Move Up** and **Move Down** buttons to adjust its sequence in the node. Make sure it is placed after the MFP Runtime Deployment component but before the MFP Server Application Adapter Deployment component.
   c. Click the newly added application or adapter component. The properties of the selected component are displayed next to the canvas. Upload the application or adapter artifact and specify its target path by referring to steps 6b-c
   d. Repeat steps 7a-c to add more applications and adapters for deployment.

7. Optional: Configure application and adapter deployment to MobileFirst Server. You can skip this step if you want to specify the user credential with deployment privilege later during the pattern deployment configuration phase in step 9. If you have specified the default admin user credential in step 3, you can now specify the deployer user, which must align with the admin user credential:
   a. In the MobileFirst Platform Server node, select the **MFP Server Application Adapter Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Find the parameters named **deployer_user** and **deployer_password**, and then click the adjacent **Delete** buttons to clear the pattern level parameter settings.
   c. In the **deployer_user** and **deployer_password** fields, specify the user name and password.
8. Configure and launch the pattern deployment:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.
   b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.
   c. In the toolbar above the panel displaying detailed information about the pattern, click the **Deploy** button.
   d. In the Deploy Pattern window, in the Configure panel, select the correct environment profile from the **Environment Profile** list, and provide other IBM PureApplication System environment parameters. To obtain the correct information, consult your IBM PureApplication System administrator.
   e. In the middle column, click **Pattern attributes** to display attributes such as user names and passwords.

      Supply the following information in the fields provided:

      Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

      `admin_user`
         Not visible if configured in step 3. Create a default MobileFirst Server administrator account. Default value: `demo`.

      `admin_password`
         Not visible if configured in step 3. Default admin account password. Default value: `demo`.

      `ACTIVATE_TOKEN_LICENSE`
         Not visible if configured in step 3. Select this field to license your pattern with token licensing. Leave this field clear if you use perpetual licenses.

      `LICENSE_SERVER_HOSTNAME`
         Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the fully-qualified hostname or IP address of your Rational License Key Server IP address. Otherwise, leave this field blank.

LMGRD_PORT
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the license manager daemon (**lmrgd**) listens for connections on. Otherwise, leave this field blank.

The default license manager daemon port is 27000.

IBMRATL_PORT
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the vendor daemon (**ibmratl**) listens for connections on. Otherwise, leave this field blank.

The default vendor daemon port is typically 27001.

runtime_contextRoot
Not visible if configured in step 5. Context root name for the MobileFirst Server runtime. The name must start with "/".

deployer_user
Not visible if configured in step 8. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when creating the default admin user for the administration service, because in this case, the only authorized user for app and adapter deployment is the default admin user.

deployer_password
Not visible if configured in step 8. User password for the user with deployment privilege.

MFP Vms Password(root)
Root password for the MobileFirst Platform Server and MobileFirst Platform DB nodes. Default value: `passw0rd`.

MFP DB Password(Instance owner)

Instance owner password for the MobileFirst Platform DB node. Default value: `passw0rd`.

f. Click **Quick Deploy** to launch your pattern deployment. After a few seconds, a message is displayed to indicate that the pattern has started to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

If you use token licensing to licenseIBM MobileFirst Platform Foundation, your pattern will fail to deploy if insufficient license tokens are available or if the license key server IP address and port were entered incorrectly.

9. Access the MobileFirst Operations Console:
   a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there. Make sure it is in Running state.
   b. Select the pattern name and expand the **Virtual machine perspective** option in the panel displaying details of the selected instance.
   c. Find the MobileFirst Server VM that has a name similar to `MobileFirst_Platform_Server.*` and make a note of its Public IP address: you need this information in the following step.

d. In the browser, open the MobileFirst Operations Console by composing its URL with one of the following formats:
   - `http://{MFP Server VM Public IP}:9080/mfpconsole`
   - `https://{MFP Server VM Public IP}:9443/mfpconsole`

e. Log in to the Console with admin user and password specified in step 3 or step 9.

Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server

You use a predefined template to deploy MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

Token licensing requirements: If you use token licensing to license IBM MobileFirst Platform Foundation, review the requirements outlined in "Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern" on page 9-48 before you continue. The deployment of this pattern fails if the license key server cannot be contacted or if insufficient license tokens are available.

About this task

Note:

Some parameters of script packages in the template have been configured with the recommended values and are not mentioned in this section. For fine-tuning purposes, see more information about all the parameters of script packages in "Script packages for MobileFirst Server" on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see "MobileFirst Platform (Liberty server farm) template" on page 9-99.

Procedure

1. Create a pattern from the predefined template:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (Liberty server farm)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
   c. In the **Name** field, provide a name for the pattern.
   d. In the **Version** field, specify the version number of the pattern.
   e. Click **Start Building**.
2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component "Default AIX add disk" to replace the "Default add disk" component in the template to support the `jfs2` file system:

   a. In the Pattern Builder, select the **MobileFirst Platform DB** node.
   b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MobileFirst Platform DB** node).
   c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
   d. Select the **Default AIX add disk** component and specify the following attributes:

      `DISK_SIZE_GB`
         Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

      `FILESYSTEM_TYPE`
         Supported file system in AIX. Default value: `jfs2`.

      `MOUNT_POINT`
         Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

      `VOLUME_GROUP`
         Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.
   e. In the MobileFirst Platform DB node, select the **Default add disk** component, and then click the bin icon to delete it.
   f. Save the pattern.
3. Optional: Configure MobileFirst Server administration. You can skip this step if you want to specify the user credential with MobileFirst Server administration privilege later during the pattern deployment configuration phase in step 9. To specify it now, complete these steps:

   Note: If you want to configure administration security with an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.
   a. In the MobileFirst Platform Server node, click the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.
   c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.
   d. If you use token licensing to license IBM MobileFirst Platform Foundation, complete the following fields. If you do not use token licensing, leave these fields blank.

      `ACTIVATE_TOKEN_LICENSE`
         Select this field to license your pattern with token licensing.

      `LICENSE_SERVER_HOSTNAME`
         Enter the fully qualified host name or IP address of your Rational License Key Server.

      `LMGRD_PORT`
         Enter the port number that the license manager daemon (**`lmrgd`**) listens for connections on. The default license manager daemon port is 27000.

IBMRATL_PORT
Enter the port number that the vendor daemon (**ibmratl**) listens for connections on. The default vendor daemon port is typically 27001.

A default administration account for MobileFirst Server is created during pattern deployment.

4. Optional: Configure MobileFirst Server runtime deployment. You can skip this step if you want to specify the context root name for the runtime later during the pattern deployment configuration phase in step 10. To specify the context root name now, complete these steps:
   a. In the MobileFirst Platform Server node, click the **MFP Server Runtime Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **runtime_contextRoot** field, click the **Delete** button to clear the pattern level parameter setting.
   c. In the **runtime_contextRoot** field, specify the runtime context root name. Note that the context root name must start with a forward slash, `/`; for example, `/HelloWorld`.
5. Upload application and adapter artifacts:

   Important: When specifying the Target path for applications and adapters, make sure all the applications and adapters are placed in the same directory. For example, if one target path is `/opt/tmp/deploy/HelloWorld-common.json`, all the other target paths should be `/opt/tmp/deploy/*`.
   a. In the MobileFirst Platform Server node, click the **MFP Server Application** or **MFP Server Adapter** component. The properties of the selected component are displayed next to the canvas.
   b. In the **Additional file** field, click the **Browse** button to locate and upload the application or adapter artifact.
   c. In the **Target path** field, specify the full path for storing the artifact including its file name; for example, `/opt/tmp/deploy/HelloWorld-common.json`.
   d. If no application or adapter is to be deployed in the pattern, remove the relevant component by clicking the **X** button inside it. To get an empty MobileFirst Operations Console deployed without any app or adapter installed, remove the MFP Server Application Adapter Deployment component by clicking the **X** button inside it.
6. Optional: Add more application or adapter artifacts for deployment:
   a. From the Assets toolbar, expand **Software Components**, and then drag and drop an **Additional file** component onto the MobileFirst Paltform Server node in the canvas. Rename it `MobileFirst App_`*`X`* or `MobileFirst Adatper_`*`X`* (where *X* is any unique number for differentiation).
   b. Hover the cursor over the newly added App or Adapter component, and then click the **Move Up** and **Move Down** buttons to adjust its sequence in the node. Make sure it is placed after the MFP Runtime Deployment component but before the MFP Server Application Adapter Deployment component.
   c. Click the newly added App or Adapter component. The properties of the selected component are displayed next to the canvas. Upload the application or adapter artifact and specify its target path by referring to steps 6b-c
   d. Repeat steps 7a-c to add more applications and adapters for deployment.

7. Optional: Configure application and adapter deployment to MobileFirst Server. You can skip this step if you want to specify the user credential with deployment privilege later during the pattern deployment configuration phase in step 10. If you have specified the default admin user credential in step 3, you can now specify the deployer user, which must align with the admin user credential:
   a. In the MobileFirst Platform Server node, click the **MFP Server Application Adapter Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Find the parameters named **deployer_user** and **deployer_password**, and then click the adjacent **Delete** buttons to clear the pattern level parameter settings.
   c. In the **deployer_user** and **deployer_password** fields, specify the user name and password.
8. Configure base scaling policy:
   a. In the **MobileFirst Platform Server** node, select the **Base Scaling Policy** component. The properties of the selected component are displayed next to the canvas.
   b. In the **Number of Instances** field, specify the number of server nodes to be instantiated during pattern deployment. The default value is 2 in the predefined template. Because dynamic scaling is not supported in this release, do not specify values in the remaining attribute fields.
9. Configure and launch the pattern deployment. Before pattern deployment, save your pattern after each modification by clicking the **Save** button in the Pattern Builder page:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.
   b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.
   c. In the toolbar above the panel displaying detailed information about the pattern, click the **Deploy** button.
   d. In the Deploy Pattern window, in the Configure panel, select the correct environment profile from the **Environment Profile** list, and provide other IBM PureApplication System environment parameters. To obtain the correct information, consult your IBM PureApplication System administrator.
   e. In the middle column, click **Pattern attributes** to display attributes such as user names and passwords.

      Supply the following information in the fields provided:

      Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

      `runtime_contextRoot_list`

      Context root names of the MobileFirst Server runtimes in case multiple runtimes exist. Use a semicolon (;) to separate each runtime context roots; for example, `HelloMobileFirst;HelloWorld`

Important: **runtime_contextRoot_list** must align with the context root specified in the MFP Server Runtime Deployment node; otherwise, IHS will not be able to correctly route requests that contain the runtime context root.

admin_user
Not visible if configured in step 3. Create a default administrator user account. Default value: demo.

admin_password
Not visible if configured in step 3. Default administrator account password. Default value: demo.

ACTIVATE_TOKEN_LICENSE
Not visible if configured in step 3. Select this field to license your pattern with token licensing. Leave this field clear if you use perpetual licenses.

LICENSE_SERVER_HOSTNAME
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the fully-qualified hostname or IP address of your Rational License Key Server IP address. Otherwise, leave this field blank.

LMGRD_PORT
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the license manager daemon (**lmrgd**) listens for connections on. Otherwise, leave this field blank.

The default license manager daemon port is 27000.

IBMRATL_PORT
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the vendor daemon (**ibmratl**) listens for connections on. Otherwise, leave this field blank.

The default vendor daemon port is typically 27001.

runtime_contextRoot
Not visible if configured in step 5. Context root name for the runtime. The name must start with /.

deployer_user
Not visible if configured in step 8. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when creating the default admin user for the administration service, because in this case, the only authorized user for app and adapter deployment is the default admin user.

deployer_password
Not visible if configured in step 8. User password for the user with deployment privilege.

MFP Vms Password(root)
Root password for the MobileFirst Platform Server and MobileFirst Platform DB nodes. Default value: `passw0rd`.

`MFP DB Password(Instance owner)`
Instance owner password for the MobileFirst Platform DB node. Default value: `passw0rd`.

f. Click **Quick Deploy** to launch your pattern deployment. After few seconds, a message is displayed to indicate that the pattern has started to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

If you use token licensing to license IBM MobileFirst Platform Foundation, your pattern will fail to deploy if insufficient license tokens are available or if the license key server IP address and port were entered incorrectly.

10. Access the MobileFirst Operations Console:
    a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there. Make sure it is in Running state.
    b. Select the pattern name and expand the **Virtual machine perspective** option in the panel displaying details of the selected instance.
    c. Find the IHS Server VM that has a name similar to `IHS_Server.*` and make a note of its Public IP address: you need this information in the following step.
    d. In the browser, open the MobileFirst Operations Console by composing its URL with one of the following formats:
       - `http://{`*`IHS Server VM Public IP`*`}/mfpconsole`
       - `https://{`*`IHS Server VM Public IP`*`}/mfpconsole`
    e. Log in to the Console with the admin user ID and password specified in step 3 or step 10.

Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server

You use a predefined template to deploy a single-node MobileFirst Server to a WebSphere Application Server full profile server.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

Token licensing requirements: If you use token licensing to license IBM MobileFirst Platform Foundation, review the requirements outlined in "Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern" on page 9-48 before you continue. The deployment of this pattern fails if the license key server cannot be contacted or if insufficient license tokens are available.

About this task

Note:

Some parameters of script packages in the template have been configured with the recommended values and are not mentioned in this section. For fine-tuning purposes, see more information about all the parameters of script packages in "Script packages for MobileFirst Server" on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see "MobileFirst Platform (WAS single node) template" on page 9-100.

Procedure

1. Create a pattern from the predefined template:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (WAS single node)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
   c. In the **Name** field, provide a name for the pattern.
   d. In the **Version** field, specify the version number of the pattern.
   e. Click **Start Building**.
2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component "Default AIX add disk" to replace the "Default add disk" component in the template to support the `jfs2` file system:
   a. In the Pattern Builder, select the **MobileFirst Platform DB** node.
   b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MobileFirst Platform DB** node).
   c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
   d. Select the **Default AIX add disk** component and specify the following attributes:

      `DISK_SIZE_GB`
      Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

      `FILESYSTEM_TYPE`
      Supported file system in AIX. Default value: `jfs2`.

      `MOUNT_POINT`
      Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

      `VOLUME_GROUP`
      Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.
   e. In the MobileFirst Platform DB node, select the **Default add disk** component, and then click the bin icon to delete it.
   f. Save the pattern.
3. Optional: Configure MobileFirst Server administration. You can skip this step if you want to specify the user credential with MobileFirst Server administration privilege later during the pattern deployment configuration phase in step 9. To specify it now, complete these steps:

Note: If you want to configure administration security with an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

a. In the MobileFirst Platform Server node, click the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.
b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.
c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.
d. If you use token licensing to license IBM MobileFirst Platform Foundation, complete the following fields. If you do not use token licensing, leave these fields blank.

   `ACTIVATE_TOKEN_LICENSE`
   Select this field to license your pattern with token licensing.

   `LICENSE_SERVER_HOSTNAME`
   Enter the fully qualified host name or IP address of your Rational License Key Server.

   `LMGRD_PORT`
   Enter the port number that the license manager daemon (**`lmrgd`**) listens for connections on. The default license manager daemon port is 27000.

   `IBMRATL_PORT`
   Enter the port number that the vendor daemon (**`ibmratl`**) listens for connections on. The default vendor daemon port is typically 27001.

A default administration account for MobileFirst Server is created during pattern deployment.

4. Optional: Configure MobileFirst Server runtime deployment. You can skip this step if you want to specify the context root name for the runtime later during the pattern deployment configuration phase in step 9. To specify the context root name now, complete these steps:
   a. In the MobileFirst Platform Server node, click the **MFP Server Runtime Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **runtime_contextRoot** field, click the **Delete** button to clear the pattern level parameter setting.
   c. In the **runtime_contextRoot** field, specify the runtime context root name. Note that the context root name must start with a forward slash, `/`; for example, `/HelloWorld`.
5. Upload application and adapter artifacts:

   Important: When specifying the Target path for applications and adapters, make sure all the applications and adapters are placed in the same directory. For example, if one target path is `/opt/tmp/deploy/HelloWorld-common.json`, all the other target paths should be `/opt/tmp/deploy/*`.
   a. In the MobileFirst Platform Server node, click the **MFP Server Application** or **MFP Server Adapter** component. The properties of the selected component are displayed next to the canvas.
   b. In the **Additional file** field, click the **Browse** button to locate and upload the application or adapter artifact.

c. In the **Target path** field, specify the full path for storing the artifact including its file name; for example, `/opt/tmp/deploy/HelloWorld-common.json`.

d. If no application or adapter is to be deployed in the pattern, remove the relevant component by clicking the **X** button inside it. To get an empty MobileFirst Operations Console deployed without any app or adapter installed, remove the MFP Server Application Adapter Deployment component by clicking the **X** button inside it.

6. Optional: Add more application or adapter artifacts for deployment:

   a. From the **Assets** toolbar, expand **Software Components**, and then drag and drop an **Additional file** component onto the MobileFirst Paltform Server node in the canvas. Rename it `MobileFirst App_`*X* or `MobileFirst Adatper_`*X* (where *X* stands for a unique number for differentiation).

   b. Hover the cursor over the newly added App or Adapter component, and then click the **Move Up** and **Move Down** buttons to adjust its sequence in the node. Make sure it is placed after the MFP Runtime Deployment component but before the MFP Server Application Adapter Deployment component.

   c. Click the newly added application or adapter component. The properties of the selected component are displayed next to the canvas. Upload the application or adapter artifact and specify its target path by referring to steps 6b-c

   d. Repeat steps 7a-c to add more applications and adapters for deployment.

7. Optional: Configure MobileFirst Server application and adapter deployment. You can skip this step if you want to specify the user credential with deployment privilege later during the pattern deployment configuration phase in step 9. If you have specified the default admin user credential in step 3, you can now specify the deployer user, which should align with the admin user credential:

   a. In the MobileFirst Platform Server node, select **MFP Server Application Adapter Deployment**. The properties of the selected component are displayed next to the canvas.

   b. Find the parameters named **deployer_user** and **deployer_password**, and then click the adjacent **Delete** buttons to delete their Pattern Level Parameter settings.

   c. In the **deployer_user** and **deployer_password** fields, specify the user name and password.

8. Configure and launch the pattern deployment:

   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.

   b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.

   c. In the toolbar above the panel displaying detailed information about the pattern, click the **Deploy** button.

   d. In the Deploy Pattern window, in the Configure panel, select the correct Environment Profile and other IBM PureApplication System environment parameters by consulting your IBM PureApplication System administrator.

   e. In the middle column, click **Pattern attributes** to set attributes such as user name and passwords.

      Supply the following information in the fields provided:

Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

`WebSphere administrative user name`
: Admin user ID for WebSphere administration console login. Default value: `virtuser`.

`WebSphere administrative password`
: Admin user password for WebSphere administration console login. Default value: `passw0rd`.

`admin_user`
: Not visible if configured in step 3. Create a default user as MobileFirst Server administrator. Default value: `demo`.

`admin_password`
: Not visible if configured in step 3. Default admin user password. Default value: `demo`.

`ACTIVATE_TOKEN_LICENSE`
: Not visible if configured in step 3. Select this field to license your pattern with token licensing. Leave this field clear if you use perpetual licenses.

`LICENSE_SERVER_HOSTNAME`
: Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the fully-qualified hostname or IP address of your Rational License Key Server IP address. Otherwise, leave this field blank.

`LMGRD_PORT`
: Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the license manager daemon (**`lmrgd`**) listens for connections on. Otherwise, leave this field blank.

  The default license manager daemon port is 27000.

`IBMRATL_PORT`
: Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the vendor daemon (**`ibmratl`**) listens for connections on. Otherwise, leave this field blank.

  The default vendor daemon port is typically 27001.

`runtime_contextRoot`
: Not visible if configured in step 5. Context root name for the MobileFirst Server runtime. The name must start with `/`.

`deployer_user`
: Not visible if configured in step 8. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when creating the default admin user for the administration service, because in this case, the only authorized user for app and adapter deployment is the default admin user.

deployer_password
: Not visible if configured in step 8. User password for the user with deployment privilege.

MFP Vms Password(root)
: Root password for the MobileFirst Platform Server and MobileFirst Platform DB virtual machines. Default value: `passw0rd`.

MFP DB Password(Instance owner)
: Instance owner password for MobileFirst Platform DB. Default value: `passw0rd`.

Important restriction:

When you set these attrbutes, do not change the following attributes in the **MobileFirst Platform Server** section:

- Cell name
- Node name
- Profile name

If you change any of these attributes, your pattern deployment will fail.

f. Click **Quick Deploy** to launch your pattern deployment. After a few seconds, a message is displayed to indicate that the pattern has started to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

If you use token licensing to licenseIBM MobileFirst Platform Foundation, your pattern will fail to deploy if insufficient license tokens are available or if the license key server IP address and port were entered incorrectly.

9. Access the MobileFirst Operations Console:
   a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there. Make sure it is in Running state.
   b. Select the pattern name and expand the **Virtual machine perspective** option in the panel displaying details of the selected instance.
   c. Find the MobileFirst Server VM that has a name similar to `MobileFirst_Platform_Server.*` and make a note of its Public IP address: you need this information in the following step.
   d. In the browser, open the MobileFirst Operations Console by composing its URL with one of the following formats:
      - `http://{`*`MFP Server VM Public IP`*`}:9080/mfpconsole`
      - `https://{`*`MFP Server VM Public IP`*`}:9443/mfpconsole`
   e. Log in to the Console with admin user and password specified in step 3 or step 9.

Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server

You use a predefined template to deploy MobileFirst Server on a multiple-node WebSphere Application Server full profile server.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

Token licensing requirements: If you use token licensing to license IBM MobileFirst Platform Foundation, review the requirements outlined in "Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern" on page 9-48 before you continue. The deployment of this pattern fails if the license key server cannot be contacted or if insufficient license tokens are available.

About this task

Note:

Some parameters of script packages in the template have been configured with the recommended values and are not mentioned in this section. For fine-tuning purposes, see more information about all the parameters of script packages in "Script packages for MobileFirst Server" on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see "MobileFirst Platform (WAS server farm) template" on page 9-102.

Procedure

1. Create a pattern from the predefined template:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (WAS server farm)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
   c. In the **Name** field, provide a name for the pattern.
   d. In the **Version** field, specify the version number of the pattern.
   e. Click **Start Building**.
2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component "Default AIX add disk" to replace the "Default add disk" component in the template to support the `jfs2` file system:
   a. In the Pattern Builder, select the **MobileFirst Platform DB** node.
   b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MobileFirst Platform DB** node).
   c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
   d. Select the **Default AIX add disk** component and specify the following attributes:

      `DISK_SIZE_GB`
      Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

FILESYSTEM_TYPE
: Supported file system in AIX. Default value: `jfs2`.

MOUNT_POINT
: Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

VOLUME_GROUP
: Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.

e. In the MobileFirst Platform DB node, select the **Default add disk** component, and then click the bin icon to delete it.

f. Save the pattern.

3. Optional: Configure MobileFirst Server administration. You can skip this step if you want to specify the user credential with MobileFirst Server administration privilege later during the pattern deployment configuration phase in step 9. To specify it now, complete these steps:

   Note: If you want to configure administration security with an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

   a. In the MobileFirst Platform Server node, click the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.
   c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.
   d. If you use token licensing to license IBM MobileFirst Platform Foundation, complete the following fields. If you do not use token licensing, leave these fields blank.

      ACTIVATE_TOKEN_LICENSE
      : Select this field to license your pattern with token licensing.

      LICENSE_SERVER_HOSTNAME
      : Enter the fully qualified host name or IP address of your Rational License Key Server.

      LMGRD_PORT
      : Enter the port number that the license manager daemon (**`lmrgd`**) listens for connections on. The default license manager daemon port is 27000.

      IBMRATL_PORT
      : Enter the port number that the vendor daemon (**`ibmratl`**) listens for connections on. The default vendor daemon port is typically 27001.

   A default administration account for MobileFirst Server is created during pattern deployment.

4. Optional: Configure MobileFirst Server runtime deployment. You can skip this step if you want to specify the context root name for the runtime later during the pattern deployment configuration phase in step 10. To specify the context root name now, complete these steps:

   a. In the MobileFirst Platform Server node, click the **MFP Server Runtime Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **runtime_contextRoot** field, click the **Delete** button to clear the pattern level parameter setting.
   c. In the **runtime_contextRoot** field, specify the runtime context root name. Note that the context root name must start with a forward slash, `/`; for example, `/HelloWorld`.
5. Upload application and adapter artifacts:

   Important: When specifying the Target path for applications and adapters, make sure all the applications and adapters are placed in the same directory. For example, if one target path is `/opt/tmp/deploy/HelloWorld-common.json`, all the other target paths should be `/opt/tmp/deploy/*`.
   a. In the MobileFirst Platform Server node, click the **MFP Server Application** or **MFP Server Adapter** component. The properties of the selected component are displayed next to the canvas.
   b. In the **Additional file** field, click the **Browse** button to locate and upload the application or adapter artifact.
   c. In the **Target path** field, specify the full path for storing the artifact including its file name; for example, `/opt/tmp/deploy/HelloWorld-common.json`.
   d. If no application or adapter is to be deployed in the pattern, remove the relevant component by clicking the **X** button inside it. To get an empty MobileFirst Operations Console deployed without any app or adapter installed, remove the MFP Server Application Adapter Deployment component by clicking the **X** button inside it.
6. Optional: Add more application or adapter artifacts for deployment:
   a. From the **Assets** toolbar, expand **Software Components**, and then drag and drop an **Additional file** component onto the MobileFirst Paltform Server node in the canvas. Rename it `MobileFirst App_`*`X`* or `MobileFirst Adatper_`*`X`* (where *X* stands for a unique number for differentiation).
   b. Hover the cursor over the newly added App or Adapter component, and then click the **Move Up** and **Move Down** buttons to adjust its sequence in the node. Make sure it is placed after the MFP Runtime Deployment component but before the MFP Server Application Adapter Deployment component.
   c. Click the newly added App or Adapter component. The properties of the selected component are displayed next to the canvas. Upload the application or adapter artifact and specify its target path by referring to steps 6b-c.
   d. Repeat steps 7a-c to add more applications and adapters for deployment.
7. Optional: Configure MobileFirst Server application and adapter deployment. You can skip this step if you want to specify the user credential with deployment privilege later during the pattern deployment configuration phase in step 10. If you have specified the default admin user credential in step 3, you can now specify the deployer user, which should align with the admin user credential:
   a. In the MobileFirst Platform Server node, select the **MFP Server Application Adapter Deployment** component. The properties of the selected component are displayed next to the canvas.

b. Find the parameters named **deployer_user** and **deployer_password**, and then click the **Delete** buttons to clear the Pattern Level Parameter settings.

c. In the **deployer_user** and **deployer_password** fields, specify the user name and password.

8. Configure base scaling policy:

   a. In the **MobileFirst Platform Server** node, select the **Base Scaling Policy** component. The properties of the selected component are displayed next to the canvas.

   b. In the **Number of Instances** field, specify the number of server nodes to be instantiated during pattern deployment. The default value is 2 in the predefined template. Because dynamic scaling is not supported in this release, do not specify values in the remaining attribute fields.

9. Configure and launch the pattern deployment:

   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.

   b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.

   c. In the toolbar above the panel displaying detailed information about the pattern, click the **Deploy** button.

   d. In the Deploy Pattern window, in the Configure panel, select the correct environment profile from the **Environment Profile** list, and provide other IBM PureApplication System environment parameters. To obtain the correct information, consult your IBM PureApplication System administrator.

   e. In the middle column, click **Pattern attributes** to display attributes such as user names and passwords.

      Supply the following information in the fields provided:

      Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

      `runtime_contextRoot_list`

      Context root names of the MobileFirst Server runtimes in case multiple runtimes exist. Use a semicolon, ";" to separate each runtime context root; for example, `HelloMobileFirst;HelloWorld`

      Important: `runtime_contextRoot_list` must align with the context root specified in the MFP Server Runtime Deployment node; otherwise, IHS will not be able to correctly route requests that contain the runtime context root.

      `WebSphere administrative user name`
      Admin user ID for WebSphere administration console login. Default value: `virtuser`.

      `WebSphere administrative password`
      Admin user password for WebSphere administration console login. Default value: `passw0rd`.

admin_user
Not visible if configured in step 3. Create a default user as MobileFirst Server administrator. Default value: demo.

admin_password
Not visible if configured in step 3. Default admin user password. Default value: demo.

ACTIVATE_TOKEN_LICENSE
Not visible if configured in step 3. Select this field to license your pattern with token licensing. Leave this field clear if you use perpetual licenses.

LICENSE_SERVER_HOSTNAME
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the fully-qualified hostname or IP address of your Rational License Key Server IP address. Otherwise, leave this field blank.

LMGRD_PORT
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the license manager daemon (**lmrgd**) listens for connections on. Otherwise, leave this field blank.

The default license manager daemon port is 27000.

IBMRATL_PORT
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the vendor daemon (**ibmratl**) listens for connections on. Otherwise, leave this field blank.

The default vendor daemon port is typically 27001.

runtime_contextRoot
Not visible if configured in step 5. Context root name for the MobileFirst Server runtime. The name must start with a forward slash, /.

deployer_user
Not visible if configured in step 8. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when creating the default admin user for the administration service, because in this case, the only authorized user for app and adapter deployment is the default admin user.

deployer_password
Not visible if configured in step 8. User password for the user with deployment privilege.

MFP Vms Password(root)
Root password for the MobileFirst Platform Server and MobileFirst Platform DB nodes. Default value: passw0rd.

MFP DB Password(Instance owner)
Instance owner password for the MobileFirst Platform DB node. Default value: passw0rd.

Important restriction:

When you set these attrbutes, do not change the following attributes in the **MobileFirst Platform Server** section:

- Cell name
- Node name
- Profile name

If you change any of these attributes, your pattern deployment will fail.

f. Click **Quick Deploy** to launch your pattern deployment. After a few seconds, a message is displayed to indicate that the pattern has started to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

If you use token licensing to licenseIBM MobileFirst Platform Foundation, your pattern will fail to deploy if insufficient license tokens are available or if the license key server IP address and port were entered incorrectly.

10. Access the MobileFirst Operations Console:
    a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there. Make sure it is in Running state.
    b. Select the pattern name and expand the **Virtual machine perspective** option in the panel displaying details of the selected instance.
    c. Find the IHS Server VM that has a name similar to `IHS_Server.*` and make a note of its Public IP address: you need this information in the following step.
    d. In the browser, open the MobileFirst Operations Console by composing its URL with one of the following formats:
       - `http://{`*`IHS Server VM Public IP`*`}/mfpconsole`
       - `https://{`*`IHS Server VM Public IP`*`}/mfpconsole`
    e. Log in to the Console with admin user and password specified in step 3 or step 10.

Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers

You can use a predefined template to deploy MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers. This application pattern template does not support token licensing.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

If you are running the System Monitoring for WebSphere Application Server shared service, the MobileFirst runtime environment might fail to start correctly when you deploy the pattern. If possible, stop the shared service before you continue with this procedure. If you cannot stop the shared service, you might need to restart the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console to fix the problem. For more information, see “MobileFirst runtime synchronization limitation with WebSphere Application Server Network Deployment” on page 9-76.

Important token licensing restriction: This pattern template does not support token licensing. You must be using perpetual licensing when you deploy patterns based on the **MobileFirst Platform (WAS ND)** pattern template.

About this task

Note:

Some parameters of script packages in the template are configured with recommended values and are not covered in this topic. For fine-tuning purposes, see more information about all the parameters of script packages in “Script packages for MobileFirst Server” on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see “MobileFirst Platform (WAS ND) template” on page 9-104.

Procedure

1. Create a pattern from the predefined template:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (WAS ND)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
   c. In the **Name** field, provide a name for the pattern.
   d. In the **Version** field, specify the version number of the pattern.
   e. Click **Start Building**.
2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component “Default AIX add disk” to replace the “Default add disk” component in the template to support the `jfs2` file system:
   a. In the Pattern Builder, select the **MobileFirst Platform DB** node.
   b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MobileFirst Platform DB** node).
   c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
   d. Select the **Default AIX add disk** component and specify the following attributes:

      `DISK_SIZE_GB`
         Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

      `FILESYSTEM_TYPE`
         Supported file system in AIX. Default value: `jfs2`.

      `MOUNT_POINT`
         Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

VOLUME_GROUP
Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.

e. In the MobileFirst Platform DB node, select the **Default add disk** component, and then click the **X** button to delete it.
f. Save the pattern.

3. Optional: Configure MobileFirst Server administration. You can skip this step if you want to specify the user credential with MobileFirst Server administration privilege later during the pattern deployment configuration phase in step 9 on page 9-74. To specify it now, complete these steps:

   Note: If you want to configure administration security with an LDAP server, you need to supply extra LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

   a. In the DmgrNode node, click the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.
   c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.

   A default administration account for MobileFirst Server is created during pattern deployment.
4. Optional: Configure MobileFirst Server runtime deployment. You can skip this step if you want to specify the context root name for the runtime later during the pattern deployment configuration phase in step 9 on page 9-74. To specify the context root name now, complete these steps:
   a. In the DmgrNode node, click the **MFP Server Runtime Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **runtime_contextRoot** field, click the **Delete** button to clear the pattern level parameter setting.
   c. In the **runtime_contextRoot** field, specify the runtime context root name. The context root name must start with a forward slash (/). For example, `/HelloWorld`.
5. Optional: Adjust the number of application server nodes in your WebSphere Application Server Network Deployment clusters for the MobileFirst Administration component and the MobileFirst runtime environment.

   By default, the Administration component and runtime environment each have two application server nodes in their respective clusters.
   a. In the DmgrNode node, click the **MFP Server Administration** component. The properties of the component are displayed next to the canvas.
   b. In the **NUMBER_OF_CLUSTERMEMBERS** field, specify the number of application server nodes that you want in your WebSphere Application Server Network Deployment cluster for the MobileFirst Administration component.
   c. In the DmgrNode node, click the **MFP Server Runtime Deployment** component. The properties of the component are displayed next to the canvas.

d. In the **NUMBER_OF_CLUSTERMEMBERS** field, specify the number of application server nodes that you want in your WebSphere Application Server Network Deployment cluster for the MobileFirst runtime environment.

e. In the **CustomNode** node, click the **Base Scaling Policy** component.

f. Adjust the **Number of Instances** value to account for the total number of application server nodes that you entered in the **NUMBER_OF_CLUSTERMEMBERS** field for each component.

   The minimum value for **Number of Instances** is the total number of server nodes for the MobileFirst Administration component and the MobileFirst runtime environments.

   For example, the default value for **Number of Instances** is `4` for the default topology with two nodes for the administration component and two nodes for the runtime environment. If you change **NUMBER_OF_CLUSTERMEMBERS** values for the administration component to `3` and for the runtime environment to `5`, the minimum value for **Number of Instances** is `8`.

6. Upload application and adapter:

   Important: When you specify the target path for applications and adapters, make sure all the applications and adapters are placed in the same directory. For example, if one target path is `/opt/tmp/deploy/HelloWorld-common.wlapp`, all the other target paths should be `/opt/tmp/deploy/*`.

   a. In the DmgrNode node, click the **MFP Application** or **MFP Adapter** component. The properties of the selected component are displayed next to the canvas.

   b. In the **Additional file** field, click the **Browse** button to locate and upload the application or adapter artifact.

   c. In the **Target path** field, specify the full path for storing the artifact, including its file name. For example, `/opt/tmp/deploy/HelloWorld-common.wlapp`.

   d. If no application or adapter is to be deployed in the pattern, remove the relevant component by clicking the **X** button inside it. To get an empty MobileFirst Operations Console deployed without any app or adapter installed, remove the MFP Server Application Adapter Deployment component by clicking the **X** button inside it.

7. Optional: Add more application or adapter artifacts for deployment:

   a. From the **COMPONENTS** toolbar, expand **Software Components**, and then drag and drop an **Additional file** component onto the DmgrNode node in the canvas. Rename it `MobileFirst App_`*`X`* or `MobileFirst Adatper_`*`X`* (where *X* stands for a unique number for differentiation).

   b. Hover the cursor over the newly added App or Adapter component, and then click the **Move Up** and **Move Down** buttons to adjust its sequence in the node. Make sure that it is placed after the MFP Runtime Deployment component but before the MFP Server Application Adapter Deployment component.

   c. Click the newly added application or adapter component. The properties of the selected component are displayed next to the canvas.

   d. In the **Additional file** field, click the **Browse** button to locate and upload the application or adapter artifact.

   e. In the **Target path** field, specify the full path for storing the artifact, including its file name. For example, `/opt/tmp/deploy/HelloWorld-common.wlapp`.

   Repeat this step if you want to add more applications and adapters for deployment.

8. Optional: Configure application and adapter deployment to MobileFirst Server. You can skip this step if you want to specify the user credential with deployment privilege later during the pattern deployment configuration phase in step 9. If you specified the default administrative user credential in step 3 on page 9-72, you can now specify the deployer user, which must align with the administration user credential:
   a. In the DmgrNode node, select the **MFP Server Application Adapter Deployment** component. The properties of the selected component are displayed next to the canvas.
   b. Find the parameters that are named **deployer_user** and **deployer_password**, and then click the adjacent **Delete** buttons to clear the pattern level parameter settings.
   c. In the **deployer_user** and **deployer_password** fields, specify the user name and password.
9. Configure and launch the pattern deployment:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.
   b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.
   c. In the toolbar above the panel that displays detailed information about the pattern, click the **Deploy** button.
   d. In the Deploy Pattern window, in the Configure panel, select the correct environment profile from the **Environment Profile** list, and provide other IBM PureApplication System environment parameters. To obtain the correct information, consult your IBM PureApplication system administrator.
   e. In the middle column, click **Pattern attributes** to display attributes such as user names and passwords.

      Supply the following information in the fields provided:

      Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply extra LDAP information. For more information, see “Configuring MobileFirst administration security with an external LDAP repository” on page 9-86.

      `WebSphere administrative user name`
      Administrative user ID for the WebSphere administrative console login. Default value: `virtuser`.

      `WebSphere administrative password`
      Administrative user password for the WebSphere administrative console login. Default value: `passw0rd`.

      `runtime_contextRoot_list`
      Context root names of the MobileFirst Server runtimes in case multiple runtimes exist. Use a semicolon (**;**) to separate each runtime context root; for example, `HelloMobileFirst;HelloWorld`

Important: This value must align with the context root specified in the MobileFirst Platform Server Runtime Deployment node that you set in the **runtime_contextRoot** field (either earlier in step 4 on page 9-72 or later in this step); otherwise, IBM HTTP Server cannot correctly route requests that contain the runtime context root.

`admin_user`
Not visible if configured in step 3 on page 9-72. Create a default MobileFirst Server administrator account. Default value: `demo`.

`admin_password`
Not visible if configured in step 3 on page 9-72. Default admin account password. Default value: `demo`.

`runtime_contextRoot`
Not visible if configured in step 4 on page 9-72. Context root name for the MobileFirst Server runtime. The name must start with "/".

`deployer_user`
Not visible if configured in step 8 on page 9-74. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when you create the default administrative user for the administration service, because in this case, the only authorized user for app and adapter deployment is the default administrative user.

`deployer_password`
Not visible if configured in step 8 on page 9-74. User password for the user with deployment privilege.

`MFP VMs Password(root)`
Root password for the DmgrNode, CustomNode, IHSNode, and MobileFirst Platform DB nodes. Default value: `passw0rd`.

`MFP VMs Password(virtuser)`

Password for the virtuser user of the DmgrNode, CustomNode, IHSNode and MobileFirst Platform DB nodes. Default value: `passw0rd`.

`Open firewall ports for WAS`
The WebSphere Application Server nodes that are deployed in the CustomNode VM nodes require open firewall ports to connect to the database server and the LDAP server (if configured for LDAP). If you need to specify multiple port numbers, separate the port numbers with a semicolon (`;`). For example, `50000;636`The default value is `50000` (the default port for DB2 server).

Important restriction:

When you set these attrbutes, do not change the following attributes in the **DmgrNode** or **CustomNode** sections:
- Cell name
- Node name
- Profile name

If you change any of these attributes, your pattern deployment will fail.

f. Click **Quick Deploy** to launch your pattern deployment. After a few seconds, a message is displayed to indicate that the pattern started to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** open the Virtual System Instances page and search for your pattern there.

10. Access the MobileFirst Operations Console:
    a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there. Make sure that it is in “Running” state.
    b. Select the pattern name and expand the **Virtual machine perspective** option in the panel that displays details of the selected instance.
    c. Find the IHS Server VM that has a name similar to `IHS_Server.*` and make a note of its Public IP address: you need this information in the following step.
    d. In the browser, open the MobileFirst Operations Console with one of the following URLs:
       - `http://{`*`IHS Server VM Public IP`*`}:80/mfpconsole`
       - `https://{`*`IHS Server VM Public IP`*`}:443/mfpconsole`
    e. Log in to the Console with admin user and password that you specified in step 3 on page 9-72 or step 9 on page 9-74.

       If the console does not display the MobileFirst runtimes, restart the IBM MobileFirst Platform runtime node from the WebSphere Application Server administrative console. For instructions about restarting the runtime node from the administrative console, see “Restarting the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console” on page 9-77.

Related concepts:

“MobileFirst runtime synchronization limitation with WebSphere Application Server Network Deployment”
If you deploy a PureApplication pattern based on the **MobileFirst Platform (WAS ND)** template and run the System Monitoring for WebSphere Application Server shared service, the MobileFirst runtime environment might fail to start correctly, when you deploy the pattern.

Related tasks:

“Restarting the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console” on page 9-77
If your MobileFirst Operations Console is empty after you deploy a PureApplication System pattern based on the **MobileFirst Platform (WAS ND)** template, you might need to restart the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console.

MobileFirst runtime synchronization limitation with WebSphere Application Server Network Deployment

If you deploy a PureApplication pattern based on the **MobileFirst Platform (WAS ND)** template and run the System Monitoring for WebSphere Application Server shared service, the MobileFirst runtime environment might fail to start correctly, when you deploy the pattern.

A PureApplication virtual system pattern based on the **MobileFirst Platform (WAS ND)** template deploys the MobileFirst administration service and the IBM MobileFirst Platform runtime into different WebSphere Application Server Network Deployment clusters. For the IBM MobileFirst Platform runtime to work correctly, it must be started after the MobileFirst administration service. If the IBM MobileFirst Platform runtime starts first, the runtime service fails to detect the MobileFirst administration service, which causes errors in the runtime service.

When the deployment of a PureApplication pattern is almost complete, the System Monitoring for WebSphere Application Server shared service restarts all of the WebSphere Application Server nodes that are deployed from the pattern. The nodes restart in a random order, so the nodes that contain the IBM MobileFirst Platform runtime might be restarted before the nodes that contain the MobileFirst administration service.

You must stop the System Monitoring for WebSphere Application Server shared service before you deploy the pattern. If you cannot stop the shared service, you might need to restart the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console to fix the problem.

Related tasks:

"Restarting the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console"
If your MobileFirst Operations Console is empty after you deploy a PureApplication System pattern based on the **MobileFirst Platform (WAS ND)** template, you might need to restart the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console.

"Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70
You can use a predefined template to deploy MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers. This application pattern template does not support token licensing.

Restarting the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console

If your MobileFirst Operations Console is empty after you deploy a PureApplication System pattern based on the **MobileFirst Platform (WAS ND)** template, you might need to restart the IBM MobileFirst Platform runtime from the WebSphere Application Server administrative console.

Before you begin

This procedure applies only when you are deploying PureApplication virtual system patterns based on the **MobileFirst Platform (WAS ND)** template when you are running the System Monitoring for WebSphere Application Server shared service. If you do not use this shared service or are deploying a pattern based on a different template, this procedure does not apply to you.

You must deploy your pattern before you do this procedure.

About this task

To work correctly, the MobileFirst administration service nodes must be started before the IBM MobileFirst Platform runtime nodes. If the System Monitoring for WebSphere Application Server shared service is running when you deploy a pattern, the shared service restarts all of the WebSphere Application Server nodes that are deployed from the pattern. The nodes restart in a random order, which means that the IBM MobileFirst Platform runtime nodes might be started before the MobileFirst administration service nodes.

Procedure

1. Confirm that the System Monitoring for WebSphere Application Server shared service is deployed and running:

a. In the PureApplication System dashboard, click **Patterns** and then under **Pattern Instances**, click **Shared Services**.

   Important: Shared Services appears twice in the **Patterns** menu, ensure that you click **Shared Services** under **Pattern Instances** and not under **Patterns.**.

b. On the Shared Service Instances page, look for a name that starts with **System Monitoring for WebSphere Application Server**. Click that name to expand its entry

   If you do not see an entry for **System Monitoring for WebSphere Application Server**, the System Monitoring for WebSphere Application Server shared service is not deployed and you do not need to continue with this procedure.

c. Check the **Status** column for the service.

   If **Status** says `Stopped`, the System Monitoring for WebSphere Application Server shared service is stopped and you do not need to continue with this procedure. If **Status** says `Started`, the System Monitoring for WebSphere Application Server shared service is running. Continue with the rest of this procedure.

2. Confirm that your pattern is running, and access the MobileFirst Operations Console from the PureApplication System dashboard.

   For instructions about how access the MobileFirst Operations Console from the PureApplication System dashboard, see step 10 on page 9-76 in "Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70.

3. If the console appears empty or is otherwise not displaying MobileFirst runtimes, restart the IBM MobileFirst Platform runtime node from the WebSphere Application Server administrative console:

   a. In the PureApplication System dashboard, click **Patterns** > **Virtual System Instances**.

   b. On the Virtual System Instances page, find your pattern instance and confirm that it is running. If it is not running, start the pattern instance.

   c. Click the name of your pattern instance and in the details panel, find the **Virtual machine perspective** section.

   d. In the Virtual machine perspective section, find the virtual machine whose name starts with `DmgrNode` and note its public IP address.

   e. Open the WebSphere Application Server administrative console at the following URL:

      `https://{`*`DmgrNode VM public IP address`*`}:9043/ibm/console`

      Use the user ID and password that you specified for the WebSphere Application Server administrative console when you deployed the pattern.

   f. In the WebSphere Application Server administrative console, expand **Applications** and click **All applications**.

   g. Restart the IBM MobileFirst Platform runtime:

      1) In the list of applications, select the application with name that begins with `IBM_Worklight_project_runtime_MFP`.
      2) In the **Action** column, select **Stop**.
      3) Click **Submit Action**.
      4) Wait until the application status in the **Status** column shows the stopped icon.
      5) In the **Action** column, select **Start**.

6) Click **Submit Action**.

Repeat this step for each IBM MobileFirst Platform runtime application in the list.

4. Access the MobileFirst Operations Console again and confirm that your IBM MobileFirst Platform runtimes are now visible.

Related concepts:

"MobileFirst runtime synchronization limitation with WebSphere Application Server Network Deployment" on page 9-76
If you deploy a PureApplication pattern based on the **MobileFirst Platform (WAS ND)** template and run the System Monitoring for WebSphere Application Server shared service, the MobileFirst runtime environment might fail to start correctly, when you deploy the pattern.

Related tasks:

"Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70
You can use a predefined template to deploy MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers. This application pattern template does not support token licensing.

Deploying MobileFirst Application Center on a single-node WebSphere Application Server Liberty profile server

You use a predefined template to deploy MobileFirst Application Center on a single-node WebSphere Application Server Liberty profile server.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

Token licensing requirements: If you use token licensing to license IBM MobileFirst Platform Foundation, review the requirements that are outlined in "Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern" on page 9-48 before you continue. If the license key server cannot be contacted or if insufficient license tokens are available then the deployment of this pattern fails .

About this task

Note:

Some parameters of script packages in the template is configured with the recommended values and are not mentioned here. For fine-tuning purposes, see more information about all the parameters of script packages in "Script packages for MobileFirst Server" on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see "MobileFirst Platform Application Center (Liberty single node) template" on page 9-106.

Procedure

1. Create a pattern from the predefined template:

a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (AppCenter Liberty single node)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
c. In the **Name** field, provide a name for the pattern.
d. In the **Version** field, specify the version number of the pattern.
e. Click **Start Building**.

2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component "Default AIX add disk" to replace the "Default add disk" component in the template to support the `jfs2` file system:
a. In the Pattern Builder, select the **MFP AppCenter DB** node.
b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MFP AppCenter DB** node).
c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
d. Select the **Default AIX add disk** component and specify the following attributes:

`DISK_SIZE_GB`
Storage size (measured in GB) to be extended to the DB server. Following is the example value: `10`.

`FILESYSTEM_TYPE`
Supported file system in AIX. Following is the default value: `jfs2`.

`MOUNT_POINT`
Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Following is the example value: `/dbinst`.

`VOLUME_GROUP`
Following is the example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.

e. In the **MFP AppCenter DB** node, select the **Default add disk** component, and then click the bin icon to delete it.
f. Save the pattern.

3. Optional: Configure **MFP Server Application Center** in the **MFP AppCenter Server** node.

Note: If you want to configure administration security with an LDAP server, you need to supply more LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

a. In the **MFP AppCenter Server** node, click the **MFP Server Application Center** component. The properties of the selected component are displayed next to the canvas.
b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.

c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.

d. Next to the **db_user** and **db_password** fields, click the **Delete** button to clear their pattern level parameter settings.

e. In the **db_user** and **db_password** fields, specify the database user name and password.

f. In the **db_name**, **db_instance**, **db_ip**, and **db_port** fields, specify the database user name, password, instance name, IP, and port number.

A default administration account for MobileFirst Server is created during pattern deployment.

4. Configure and launch the pattern deployment:

a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.

b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.

c. In the toolbar above the panel that displays the detailed information about the pattern, click the **Deploy** button.

d. In the Deploy Pattern window, in the Configure panel, select the correct environment profile from the **Environment Profile** list, and provide other IBM PureApplication System environment parameters. To obtain the correct information, consult your IBM PureApplication System administrator.

e. In the middle column, click **Pattern attributes** to display attributes such as user names and passwords.

Supply the following information in the fields provided:

Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply more LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

`admin_user`
Not visible if configured in step 3. Create a default MobileFirst Server administrator account. Following is the default value: `demo`.

`admin_password`
Not visible if configured in step 3. Default admin account password. Following is the default value: `demo`.

`ACTIVATE_TOKEN_LICENSE`
Not visible if configured in step 3. Select this field to license your pattern with token licensing. If you use perpetual licenses then leave this field clear.

`LICENSE_SERVER_HOSTNAME`
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the fully qualified hostname or IP address of your Rational License Key Server IP address. Otherwise, leave this field blank.

`LMGRD_PORT`
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the license manager daemon (**`lmgrd`**) listens for connections on. Otherwise, leave this field blank.

The default license manager daemon port is 27000.

`IBMRATL_PORT`
Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the vendor daemon (**`ibmratl`**) listens for connections on. Otherwise, leave this field blank.

The default vendor daemon port is typically 27001.

`runtime_contextRoot`
Not visible if configured in step 5. Context root name for the MobileFirst Server runtime. The name must start with "/".

`deployer_user`
Not visible if configured in step 8. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when creating the default admin user for the administration service because in this case, the only authorized user for app and adapter deployment is the default admin user.

`deployer_password`
Not visible if configured in step 8. User password for the user with deployment privilege.

`MFP Vms Password(root)`
Root password for the MobileFirst Platform Server and MobileFirst Platform DB nodes. Following is the default value: `passw0rd`.

`MFP DB Password(Instance owner)`

Instance owner password for the MobileFirst Platform DB node. Following is the default value: `passw0rd`.

f. Click **Quick Deploy** to launch your pattern deployment. After a few seconds, a message is displayed to indicate that the pattern is starting to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

5. To access the MobileFirst Operations Console perform the following steps:
   a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.
   b. Select your pattern name and expand the **Virtual machine perspective** in the panel that displays the details of the selected instance.
   c. Find the MobileFirst Server VM that has a name similar to `MFP_AppCenter_Server.*`, make a note of its public IP address.
   d. In the browser, open the MobileFirst Operations Console by composing its URL with one of the following formats:
      - `http://{`*`MFP Server VM Public IP`*`}:9080/appcenterconsole`
      - `https://{`*`MFP Server VM Public IP`*`}:9443/appcenterconsole`
   e. Log in to the Console with admin user and password specified in step 3.

Deploying MobileFirst Application Center on a single-node WebSphere Application Server full profile server

You use a predefined template to deploy a single-node MobileFirst Application Center to a WebSphere Application Server full profile server.

Before you begin

This procedure involves uploading certain artifacts to IBM PureApplication System such as the required application and adapter. Before you begin, ensure that the artifacts are available for upload.

Token licensing requirements: If you use token licensing to license IBM MobileFirst Platform Foundation, review the requirements outlined in "Token licensing requirements for IBM MobileFirst Platform Foundation System Pattern" on page 9-48 before you continue. The deployment of this pattern fails if the license key server cannot be contacted or if insufficient license tokens are available.

About this task

Note:

Some parameters of script packages in the template have been configured with the recommended values and are not mentioned in this section. For fine-tuning purposes, see more information about all the parameters of script packages in "Script packages for MobileFirst Server" on page 9-108.

For more information about the composition and configuration options of the predefined template that is used in this procedure, see "MobileFirst Platform Application Center (WAS single node) template" on page 9-107.

Procedure

1. Create a pattern from the predefined template:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, click **Create New**, and then in the pop-up window, select **MobileFirst Platform (AppCenter Liberty single node)** from the list of predefined templates. If the name is only partially visible due to its length, you can confirm that the correct template is selected by viewing its description on the **More information** tab.
   c. In the **Name** field, provide a name for the pattern.
   d. In the **Version** field, specify the version number of the pattern.
   e. Click **Start Building**.
2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component "Default AIX add disk" to replace the "Default add disk" component in the template to support the `jfs2` file system:
   a. In the Pattern Builder, select the **MFP AppCenter DB** node.
   b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MFP AppCenter DB** node).
   c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
   d. Select the **Default AIX add disk** component and specify the following attributes:

      `DISK_SIZE_GB`
      Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

FILESYSTEM_TYPE
Supported file system in AIX. Default value: `jfs2`.

MOUNT_POINT
Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

VOLUME_GROUP
Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.

e. In the **MFP AppCenter DB** node, select the **Default add disk** component, and then click the bin icon to delete it.

f. Save the pattern.

3. Optional: Configure **MFP Server Application Center** in the **MFP AppCenter Server** node.

   Note: If you want to configure administration security with an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

   a. In the **MFP AppCenter Server** node, click the **MFP Server Application Center** component. The properties of the selected component are displayed next to the canvas.
   b. Next to the **admin_user** and **admin_password** fields, click the **Delete** button to clear their pattern level parameter settings.
   c. In the **admin_user** and **admin_password** fields, specify the administration user name and password.
   d. Next to the **db_user** and **db_password** fields, click the **Delete** button to clear their pattern level parameter settings.
   e. In the **db_user** and **db_password** fields, specify the database user name and password.
   f. In the **db_name**, **db_instance**, **db_ip**, and **db_port** fields, specify the database user name, password, instance name, IP, and port number.

   A default administration account for MobileFirst Server is created during pattern deployment.

4. Configure and launch the pattern deployment:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**.
   b. On the Virtual System Patterns page, use the **Search** field to find the pattern you created, and then select the pattern.
   c. In the toolbar above the panel displaying detailed information about the pattern, click the **Deploy** button.
   d. In the Deploy Pattern window, in the Configure panel, select the correct environment profile from the **Environment Profile** list, and provide other IBM PureApplication System environment parameters. To obtain the correct information, consult your IBM PureApplication System administrator.
   e. In the middle column, click **Pattern attributes** to display attributes such as user names and passwords.

      Supply the following information in the fields provided:

Note: Make appropriate changes to the default values of the pattern-level parameters even if an external LDAP server is configured. If you configure administration security by using an LDAP server, you need to supply additional LDAP information. For more information, see "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

`admin_user`
: Not visible if configured in step 3. Create a default MobileFirst Server administrator account. Default value: `demo`.

`admin_password`
: Not visible if configured in step 3. Default admin account password. Default value: `demo`.

`ACTIVATE_TOKEN_LICENSE`
: Not visible if configured in step 3. Select this field to license your pattern with token licensing. Leave this field clear if you use perpetual licenses.

`LICENSE_SERVER_HOSTNAME`
: Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the fully-qualified hostname or IP address of your Rational License Key Server IP address. Otherwise, leave this field blank.

`LMGRD_PORT`
: Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the license manager daemon (**`lmrgd`**) listens for connections on. Otherwise, leave this field blank.

  The default license manager daemon port is 27000.

`IBMRATL_PORT`
: Not visible if configured in step 3. If you use token licensing to license IBM MobileFirst Platform Foundation, enter the port number that the vendor daemon (**`ibmratl`**) listens for connections on. Otherwise, leave this field blank.

  The default vendor daemon port is typically 27001.

`runtime_contextRoot`
: Not visible if configured in step 5. Context root name for the MobileFirst Server runtime. The name must start with "/".

`deployer_user`
: Not visible if configured in step 8. User name for the account with deployment privilege. If an external LDAP server is not configured, you must enter the same value as was specified when creating the default admin user for the administration service, because in this case, the only authorized user for app and adapter deployment is the default admin user.

`deployer_password`
: Not visible if configured in step 8. User password for the user with deployment privilege.

`MFP Vms Password(root)`
: Root password for the MobileFirst Platform Server and MobileFirst Platform DB nodes. Default value: `passw0rd`.

MFP DB Password(Instance owner)

Instance owner password for the MobileFirst Platform DB node. Default value: `passw0rd`.

f. Click **Quick Deploy** to launch your pattern deployment. After a few seconds, a message is displayed to indicate that the pattern has started to launch. You can click the URL provided in the message to track your pattern deployment status or go to **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

5. To access the MobileFirst Operations Console perform the following steps:

a. Click **Patterns** > **Virtual System Instances** to open the Virtual System Instances page and search for your pattern there.

b. Select your pattern name and expand the **Virtual machine perspective** in the panel that displays the details of the selected instance.

c. Find the MobileFirst Server VM that has a name similar to `MFP_AppCenter_Server.*`, make a note of its public IP address.

d. In the browser, open the MobileFirst Operations Console by composing its URL with one of the following formats:

- `http://{`*`MFP Server VM Public IP`*`}:9080/appcenterconsole`
- `https://{`*`MFP Server VM Public IP`*`}:9443/appcenterconsole`

e. Log in to the Console with admin user and password specified in step 3.

Configuring MobileFirst administration security with an external LDAP repository

You can configure MobileFirst administration security to enable connecting out to an external LDAP repository. The configuration is common for both WebSphere Application Server Liberty profile and full profile.

Before you begin

This procedure involves configuring the LDAP parameters for connecting to the external user registry server. Before you begin, ensure the LDAP server is working and consult your LDAP administrator to obtain the required configuration information.

About this task

Important:

When the LDAP repository configuration is enabled, a default user for MobileFirst administration is not automatically created. Instead, you must specify the administration user name and password that are stored in the LDAP repository. This information is required by WebSphere Application Server Liberty profile and a server farm of WebSphere Application Server full profile.

If the runtime to be deployed in the pattern is configured to use LDAP for application authentication, make sure that the LDAP server configured in the runtime is the same as the LDAP server that is configured for the MobileFirst Administration; different LDAP servers are not supported. Also, the protocol and port for LDAP connection must be identical. For example, if connections from the runtime to the LDAP server are configured to use the SSL protocol and port is 636, connections from the MobileFirst Administration to the LDAP server must use the SSL protocol and port 636 as well.

Procedure

1. Build a pattern with any topology you need. For more information, see the following topics:
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server" on page 9-49
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server" on page 9-54
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server" on page 9-64
   - "Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70
2. Mandatory for AIX: In IBM PureApplication System running on Power, the MobileFirst Platform DB node needs to use the AIX-specific add-on component "Default AIX add disk" to replace the "Default add disk" component in the template to support the `jfs2` file system:
   a. In the Pattern Builder, select the **MobileFirst Platform DB** node.
   b. Click the **Add a Component Add-on** button (the button is visible above the component box when you hover the cursor over the **MobileFirst Platform DB** node).
   c. From the **Add Add-ons** list, select **Default AIX add disk**. The component is added as the lowest component of the MobileFirst Platform DB node.
   d. Select the **Default AIX add disk** component and specify the following attributes:

      `DISK_SIZE_GB`
         Storage size (measured in GB) to be extended to the DB server. Example value: `10`.

      `FILESYSTEM_TYPE`
         Supported file system in AIX. Default value: `jfs2`.

      `MOUNT_POINT`
         Align with the attribute **Mount point for instance owner** in the Database Server component in the MobileFirst Platform DB node. Example value: `/dbinst`.

      `VOLUME_GROUP`
         Example value: `group1`. Contact your IBM PureApplication System administrator for the correct value.
   e. In the MobileFirst Platform DB node, select the **Default add disk** component, and then click the bin icon to delete it.
   f. Save the pattern.
3. Configure MobileFirst Server administration:
   a. In IBM PureApplication System, in the dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, use the **Search** field to find and select the pattern you created, and then click **Open** to open the Pattern Builder page.
   c. In the MobileFirst Platform Server node (or the DmgrNode node when using the MobileFirst Platform (WAS ND) template), select the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.

d. Supply the following LDAP information in the fields provided:

admin_user
User ID of the account that has MobileFirst Server administration privilege. This value is stored in the LDAP repository. Not required if the MobileFirst Server is to be deployed on a single node of WebSphere Application Server full profile.

admin_password
Admin user password. This value is stored in the LDAP repository. Not required if the MobileFirst Server is to be deployed on a single node of WebSphere Application Server full profile.

LDAP_TYPE
LDAP server type of your user registry. One of the following values:

None LDAP connection is disabled. When this is set, all the other LDAP parameters are treated as placeholders only.

TivoliDirectoryServer
Select this if the LDAP repository is an IBM Tivoli® Directory Server.

ActiveDirectory
Select this if the LDAP repository is a Microsoft Active Directory.

Default value: `None`.

LDAP_IP
LDAP server IP address.

LDAP_SSL_PORT
LDAP port for secure connection.

LDAP_PORT
LDAP port for non-secure connection.

BASE_DN
Base DN.

BIND_DN
Bind DN.

BIND_PASSWORD
Bind DN password.

REQUIRE_SSL
Select `true` for secure connection to the LDAP server. Default value: `false`.

USER_FILTER
LDAP user filter that applies when searching the existing user registry for users.

GROUP_FILTER
LDAP group filter that applies when searching the existing user registry for groups.

LDAP_REPOSITORY_NAME
LDAP server name.

CERT_FILE_PATH
Target path of the uploaded LDAP server certification.

mfpadmin
Admin role for MobileFirst Server. One of the following values:

None No user.

AllAuthenticatedUsers
Authenticated users

Everyone
All users.

Default value: `None`. For more information about security roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

mfpdeployer
Deployer role for MobileFirst Server. One of the following values:

None No user.

AllAuthenticatedUsers
Authenticated users

Everyone
All users.

Default value: `None`. For more information about security roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

mfpmonitor
Monitor role for MobileFirst Server. One of the following values:

None No user.

AllAuthenticatedUsers
Authenticated users

Everyone
All users.

Default value: `None`. For more information about security roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

mfpoperator
Operator role for MobileFirst Server. One of the following values:

None No user.

AllAuthenticatedUsers
Authenticated users

Everyone
All users.

Default value: `None`. For more information about security roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

4. Optional: Configure the LDAP SSL connection. This step is required only if you set **REQUIRE_SSL** to `true` in the previous step to use secure connections to the LDAP server:

a. From the Assets toolbar, expand **Software Components**, and then drag and drop an **Additional file** component onto the MobileFirst Platform Server node in the canvas. Rename the component “MobileFirst LDAP Cert”, for example.
b. Hover the cursor over the newly added component, and then click the **Move up** and **Move down** buttons to adjust the position of the component in the node. Make sure that it is placed between the MFP Server Prerequisite component and the MFP Server Administration component.
c. Click the **MobileFirst LDAP Cert** component. The properties of the selected component are displayed next to the canvas. Upload the LDAP certification artifact in the **Additional file** field by clicking the **Browse** button to locate it
d. In the **Target path** field, specify the full path for storing the artifact including its file name; for example, `/opt/tmp/tdscert.der`.
e. In the MobileFirst Platform Server node (or the DmgrNode node when using the MobileFirst Platform (WebSphere Application Server Network Deployment), select the **MFP Server Administration** component, and then click the **Add reference** button next to the **CERT_FILE_PATH** field. In the pop-up window, click the **component-level parameter** tab. From the **Component** list, select **MobileFirst LDAP Cert**. In the **Output attribute** list, select **target_path**. Click the **ADD** button to refresh the **Output value** field, and then click **OK**.

5. Configure and launch the pattern deployment. On the Deploy Pattern page, in the Nodes list, you can adjust your LDAP configurations by clicking **MobileFirst Platform Server** (or **DmgrNode** when using the MobileFirst Platform (WAS ND) template) and then expanding **MFP Server Administration**. For more information about pattern deployment, see the “Configure and launch the pattern deployment” step in one of the following topics depending on the topology you selected when creating the pattern;
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server” on page 9-49, step 8 on page 9-52
   - “Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server” on page 9-54, step 9 on page 9-57
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server” on page 9-59, step 8 on page 9-62
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server” on page 9-59, step 9 on page 9-68
   - “Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers” on page 9-70, step 9 on page 9-74 onwards.
6. Access the MobileFirst Operations Console. Use the administrator user name and password to log in to the MobileFirst Operations Console through your LDAP configuration. For more information, see the “Access the MobileFirst Operations Console:” step in one of the following topics depending on the topology you selected when creating the pattern;
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server” on page 9-49, step 9 on page 9-53
   - “Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server” on page 9-54, step 10 on page 9-59
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server” on page 9-59, step 9 on page 9-64
   - “Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server” on page 9-64, step 10 on page 9-70

- “Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers” on page 9-70, step 10 on page 9-76 onwards.

Configuring an external database with a IBM MobileFirst Platform Foundation System Pattern

You can configure IBM MobileFirst Platform Foundation System Patterns to enable connecting out to an external database. IBM DB2 is the only supported external database. The configuration is common for all the supported patterns.

Before you begin

This procedure involves configuring the external database parameters for connecting to the external database. Before you begin, ensure the following:

- Configure the external database instance on your installed IBM DB2.
- Make a note of the database instance name, database user name, database password, database host name or IP and database instance port.

About this task

Procedure

1. Build a pattern with any topology you need. For more information, see the following topics:
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server” on page 9-49
   - “Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server” on page 9-54
   - “Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server” on page 9-59
   - “Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server” on page 9-64
   - “Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers” on page 9-70
2. Select the **MobileFirst Platform DB** and click **Remove component**.
3. Configure MobileFirst Server administration:
   a. In IBM PureApplication System, in the dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, use the **Search** field to find and select the pattern you created, and then click **Open** to open the Pattern Builder page.
   c. In the MobileFirst Platform Server node (or the DmgrNode node when using the MobileFirst Platform (WAS ND) template), select the **MFP Server Administration** component. The properties of the selected component are displayed next to the canvas.
   d. Check the option **USE_EXTERNAL_DATABASE** and configure the following parameters:

      `db_instance`
         External database instance name.

      `db_user`
         External database user name.

db_name
External database name.

db_password
External database password.

db_ip
External database IP.

db_port
External database port number.

Note: If you are using the MobileFirstPlatform (WAS ND) pattern template, you will need to additionally configure the attribute **Open firewall ports for WAS** to the external database port number.

e. In the MobileFirst Platform Server node (or the DmgrNode node when using the MobileFirst Platform (WAS ND) template), select the **MFP Server Runtime Deployment** component. The properties of the selected component are displayed next to the canvas.

f. Under the **USE_EXTERNAL_DATABASE** configure the following parameters:

rtdb_instance
External database instance name.

rtdb_user
External runtime database user name.

rtdb_name
External runtime database name, which will be created.

rtdb_password
External runtime database password.

4. Configure and launch the pattern deployment. For more information about pattern deployment, see the "Configure and launch the pattern deployment" step in one of the following topics depending on the topology you selected when creating the pattern;
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server" on page 9-49, step 8 on page 9-52
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server" on page 9-54, step 9 on page 9-57
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59, step 8 on page 9-62
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59, step 9 on page 9-68
   - "Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70, step 9 on page 9-74 onwards.

Deploying and configuring MobileFirst Analytics

You can deploy and configure the MobileFirst Analytics on both WebSphere Application Server Liberty profile and full profile to enable the Analytics features in the pattern.

Before you begin

If you intend to use an LDAP repository to protect the Analytics Console, ensure that the LDAP server is working and consult your LDAP administrator to obtain the required configuration information.

About this task

Important:

When the LDAP repository configuration is enabled in the Analytics component, a default administration user is not created for MobileFirst Analytics. Instead, you must specify the administration user name and password values that are stored in the LDAP repository. These values are required to protect the Analytics Console.

Procedure

1. Build a pattern with the topology you need. For more information, see the following topics:
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server" on page 9-49
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server" on page 9-54
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server" on page 9-64
   - "Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70
2. Add and configure MobileFirst Analytics:
   a. In the IBM PureApplication System dashboard, click **Patterns** > **Virtual System Patterns**. The Virtual System Patterns page opens.
   b. On the Virtual System Patterns page, use the **Search** field to find and select the pattern you created, and then click **Open** to open the Pattern Builder page.
   c. From the Assets list, expand **Software Components**, and then drag and drop one of the following components onto the canvas:

      Liberty profile server
      Select this component if you want to deploy MobileFirst Analytics on WebSphere Application Server Liberty profile.

      Standalone server
      Select this component if you want to deploy MobileFirst Analytics on WebSphere Application Server full profile.

      A new node is created with the name "OS Node". Rename it "MobileFirst Platform Analytics".
   d. Make the following configuration changes depending on the type of application server you want to deploy Analytics to:
      - If you are deploying MobileFirst Analytics to WebSphere Application Server Liberty profile, click **Liberty profile server** in the MobileFirst Platform Analytics node. The properties of the selected component are displayed next to the canvas. In the **Configuration data location** field,

enter the path `/opt/IBM/WebSphere/Liberty` and specify the administrative user name and password. Use the default values for the other parameters.

- If you are deploying MobileFirst Analytics to WebSphere Application Server full profile, click **Standalone server** in the MobileFirst Platform Analytics node. The properties of the selected component are displayed next to the canvas. In the **Configuration data location** field, enter the path `/opt/IBM/WebSphere/AppServer/Profiles`, change **Profile name** to `AppSrv01`, and specify the administrative user name and password. Use the default values for the other parameters.

  Important: The WebSphere Application Server administrative user will be created in the WebSphere Application Server user repository. If LDAP will be configured for the Analytics server, avoid user name conflicts with the WebSphere Application Server administrative user. For example, if "user1" will be introduced by the LDAP server through its configuration, do not set "user1" as the WebSphere Application Server administrative user name.

e. From the Components list, expand **Scripts**, and then drag and drop an **MFP Server Prerequisite** component and a **MFP WAS SDK Level** component onto the MobileFirst Platform Analytics node on the canvas.

f. From the Components list, expand **Scripts**, and then drag and drop an **MFP Analytics** component onto the MobileFirst Platform Analytics node on the canvas. Make sure the MFP Analytics component is positioned after the Liberty profile server component (or the Standalone server component).

g. Supply the following MobileFirst Analytics information in the fields provided:

   The LDAP parameters are exactly the same as the MFP Server Administration parameters. For more information, see the "Configure MFP Server Administration" step in 3 on page 9-87:

   Important: For LDAP SSL connection configuration in MobileFirst Analytics, make sure that in step 4b in "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86, the dragged-in MobileFirst LDAP Cert component in the MobileFirst Platform Analytics node must be moved to between the Liberty profile server (or Stanalone server) and the MFP Analytics script package.

   `WAS_ROOT`

   - If MobileFirst Analytics is being installed on WebSphere Application Server Liberty profile, specify the installation directory of the Liberty profile for Analytics:
     1) Click the **Add reference** button next to the **WAS_ROOT** field and in the pop-up window, click the **component-level parameter** tab.
     2) In the **Component** field, select **Liberty profile server**.(it might be called **Liberty profile server_1** if the MobileFirst Server is also deployed on WebSphere Application Server Liberty profile).
     3) In the **Output attribute** field, select **install_directory**. Click the **ADD** button to refresh the **Output value** field, and then click **OK**.
   - If MobileFirst Analytics is being installed on WebSphere Application Server full profile, specify the installation directory of the WebSphere Application Server full profile for Analytics:
     1) Click the **Add reference** button next to the **WAS_ROOT** field and in the pop-up window, click the **component-level parameter** tab.

2) In the **Component** field, select **Standalone server**.(it might be called **Standalone server_1** if the MobileFirst Server is also deployed on WebSphere Application Server full profile)
3) In the **Output attribute** field, select **install_directory**. Click the **ADD** button to refresh the **Output value** field, and then click **OK**.

`HEAP_MIN_SIZE`

Applicable to WebSphere Application Server full profile only.

The amount of Analytics data that is generated is directly proportional to the amount of memory required to handle it. Set this value to allow a larger minimum heap size for WebSphere Application Server full profile. Make sure that the **Memory size** value specified in the Core OS component of the MobileFirst Platform Analytics node is larger than **`HEAP_MIN_SIZE`**. Consider setting a value equal to **`HEAP_MAX_SIZE`**

Default value: 4096 MB.

`HEAP_MAX_SIZE`

Applicable to WebSphere Application Server full profile only.

The amount of Analytics data that is generated is directly proportional to the amount of memory required to handle it. Set this value to allow a larger maximum heap size for WebSphere Application Server full profile. Make sure that the **Memory size** value specified in the Core OS component of the MobileFirst Platform Analytics node is larger than **`HEAP_MAX_SIZE`**. Consider setting a value equal to **`HEAP_MIN_SIZE`**

Default value: 4096 MB.

`WAS_admin_user`

Applicable to WebSphere Application Server full profile only.

WebSphere Application Server full profile admin user ID for the Analytics server.

1) Click the **Add reference** button next to the **WAS_admin_user** field and in the pop-up window, click the **component-level parameter** tab.
2) In the **Component** field, select **Standalone server**.(it may be called **Standalone server_1** if the MobileFirst Server is also deployed on WebSphere Application Server full profile)
3) In the **Output attribute** field, select **was_admin**. Click the **ADD** button to refresh the **Output value** field, and then click **OK**.

For Liberty profile, the default value can be used.

`WAS_admin_password`

Applicable to WebSphere Application Server full profile only.

WebSphere Application Server full profile admin user ID for the Analytics server.

1) Click the **Add reference** button next to the **WAS_admin_password** field and in the pop-up window, click the **component-level parameter** tab.
2) In the **Component** field, select **Standalone server**.(it may be called **Standalone server_1** if the MobileFirst Server is also deployed on WebSphere Application Server full profile)

3) In the **Output attribute** field, select **was_admin_password**. Click the **ADD** button to refresh the **Output value** field, and then click **OK**.

For Liberty profile, the default value can be used.

`admin_user`

- If an LDAP repository is not enabled, create a default administration user for MobileFirst Analytics console protection.
- If an LDAP repository is enabled, specify the user name that has MobileFirst Analytics administration privilege. The value is stored in the LDAP repository.

`admin_password`

- If an LDAP repository is not enabled, specify the password for the default administration user for MobileFirst Analytics console protection.
- If an LDAP repository is enabled, specify the administration user password. The value is stored in the LDAP repository.

h. Optional: Enable the LDAP repository for MobileFirst Analytics console protection. The LDAP parameters in MobileFirst Analytics are exactly the same as those for MobileFirst Server Administration. For more information, see "Configure MFP Server Administration" (step 3 on page 9-87) in "Configuring MobileFirst administration security with an external LDAP repository" on page 9-86.

3. Configure MobileFirst Server runtime deployment for MobileFirst Analytics connection:

   a. In the MobileFirst Platform Server node (or the DmgrNode node when using the MobileFirst Platform (WAS ND) template), select the **MFP Server Runtime Deployment** component.

   b. Drag a link from the MFP Server Runtime Deployment component to the Liberty profile server component or to the Standalone server component in the MobileFirst Platform Analytics node, depending on the type of application server being used. The Configure Data Dependencies pop-up window opens.

   c. Configure the data dependencies:

      1) In the Configure Data Dependencies window, clear any existing recommended data dependency entries by clicking the **X** button next to each entry.
      2) Below MFP Server Runtime Deployment component, select **analytics_ip** and below Liberty profile server or Standalone server, select **IP**.
      3) Click the **Add** button to add the new data dependency.
      4) Click **OK** to save your changes.

   The link from the MFP Server Runtime Deployment component to the Liberty profile server component (or the Standalone server component) is built.

   d. Drag another link from the MFP Server Runtime Deployment component to the MFP Analytics component in the MobileFirst Platform Analytics node. The Configure Data Dependencies pop-up window opens.

   e. Configure the data dependencies:

      1) In the Configure Data Dependencies window, clear all the recommended data dependencies entries by clicking the **X** button next to each entry.

2) Below MFP Server Runtime Deployment component, select **analytics_admin_user** and below MFP Analytics, select **admin_user**.
3) Click the **Add** button to add the new data dependency.
4) Repeat the process to configure a data dependency from **analytics_admin_password** to **admin_password**.
5) Click **OK** to save your changes.

The link from the MFP Server Runtime Deployment component to the MFP Analytics component is built.

The following figure shows an example of a MobileFirst Platform Analytics node added to a MobileFirst Platform WAS ND pattern:

Figure 9-1. MobileFirst Platform Analytics node added to a MobileFirst Platform WAS ND pattern

4. Configure and launch the pattern deployment.

   On the Deploy Pattern page, you can adjust your MobileFirst Analytics configuration settings by clicking the **MobileFirst Platform Analytics** component under the Nodes list in the middle column and then expanding **MFP Analytics**.

   For more information about pattern deployment, see the "Configure and launch the pattern deployment" step in the following topics depending on the topology you selected when creating the pattern:
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server" on page 9-49, step 8 on page 9-52
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server" on page 9-54, step 9 on page 9-57
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59, step 8 on page 9-62
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59, step 9 on page 9-68
   - "Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70, step 9 on page 9-74
5. Access MobileFirst Analytics through the MobileFirst Operations Console.

   For more information, see the "Access the MobileFirst Operations Console" step in one of the following topics depending on the topology you selected when creating the pattern:
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server Liberty profile server" on page 9-49, step 9 on page 9-53
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server Liberty profile server" on page 9-54, step 10 on page 9-59
   - "Deploying MobileFirst Server on a single-node WebSphere Application Server full profile server" on page 9-59, step 9 on page 9-64
   - "Deploying MobileFirst Server on a multiple-node WebSphere Application Server full profile server" on page 9-64, step 10 on page 9-70
   - "Deploying MobileFirst Server on clusters of WebSphere Application Server Network Deployment servers" on page 9-70, step 10 on page 9-76

Predefined templates for MobileFirst Platform Pattern

IBM MobileFirst Platform Foundation System Pattern includes predefined templates that you can use to build patterns for the most typical deployment topologies.

The following templates are available:
- "MobileFirst Platform (Liberty single node) template"
- "MobileFirst Platform (Liberty server farm) template" on page 9-99
- "MobileFirst Platform (WAS single node) template" on page 9-100
- "MobileFirst Platform (WAS server farm) template" on page 9-102
- "MobileFirst Platform (WAS ND) template" on page 9-104
- "MobileFirst Platform Application Center (Liberty single node) template" on page 9-106
- "MobileFirst Platform Application Center (WAS single node) template" on page 9-107

MobileFirst Platform (Liberty single node) template

Figure 9-2 shows the composition of the "MobileFirst Platform (Liberty single node)" template.

Figure 9-2. MobileFirst Platform (Liberty single node) template

The "MobileFirst Platform (Liberty single node)" template is composed of the following nodes and components:

Table 9-33. MobileFirst Platform (Liberty single node) template nodes and components.

Node	Components
MobileFirst Platform Server	**Liberty profile server** WebSphere Application Server Liberty profile server installation. **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Administration** MobileFirst Server Administration web application including MobileFirst Operations Console. **MFP Server Runtime Deployment** Runtime context root configuration. **MFP Server Application** MobileFirst application to be added to the deployment. **MFP Server Adapter** MobileFirst adapter to be added to the deployment. **MFP Server Application Adapter Deployment** Application and adapter deployment to the MobileFirst Server.
MobileFirst Platform DB	**Database Server** DB2 database server installation. **MFP Administration DB** MobileFirst administration database schema installation. **MFP Runtime DB** MobileFirst runtime database schema installation. **Default add disk** Disk size configuration.

MobileFirst Platform (Liberty server farm) template

Figure 9-3 shows the composition of the "MobileFirst Platform (Liberty server farm)" template.

Figure 9-3. MobileFirst Platform (Liberty server farm) template

The "MobileFirst Platform (Liberty server farm)" template is composed of the following nodes and components:

Table 9-34. MobileFirst Platform (Liberty server farm) template nodes and components.

Node	Components
IHS Server	**IBM HTTP servers** IBM HTTP Server installation. **MFP IHS Configuration** Automatic configuration of IBM HTTP Server.
MobileFirst Platform Server	**Liberty profile server** WebSphere Application Server Liberty profile server installation. **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Administration** MobileFirst Server Administration web application including MobileFirst Operations Console. **MFP Server Runtime Deployment** Runtime context root configuration. **MFP Server Application** MobileFirst application to be added to the deployment. **MFP Server Adapter** MobileFirst adapter to be added to the deployment. **MFP Server Application Adapter Deployment** Application and adapter deployment to the MobileFirst Server. **Base Scaling Policy** VM scaling policy: number of VMs.
MobileFirst Platform DB	**Database Server** DB2 database server installation. **MFP Administration DB** MobileFirst administration database schema installation. **MFP Runtime DB** MobileFirst runtime database schema installation. **Default add disk** Disk size configuration.

MobileFirst Platform (WAS single node) template

Figure 9-4 on page 9-101 shows the composition of the “MobileFirst Platform (WAS single node)” template.

Figure 9-4. MobileFirst Platform (WAS single node) template

The "MobileFirst Platform (WAS single node)" template is composed of the following nodes and components:

Table 9-35. MobileFirst Platform (WAS single node) template nodes and components.

Node	Components
MobileFirst Platform Server	**Standalone server** WebSphere Application Server full profile server installation. **Restriction:** Do not change the values for the following component attributes: • Cell name • Node name • Profile name If you change any of these attributes, the deployment of patterns that are based on this template fails. **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Administration** MobileFirst Server Administration web application including MobileFirst Operations Console. **MFP Server Runtime Deployment** Runtime context root configuration. **MobileFirst App** MobileFirst application to be added to the deployment. **MobileFirst Adapter** MobileFirst adapter to be added to the deployment. **MFP Server Application Adapter Deployment** Application and adapter deployment to the MobileFirst Server.

Table 9-35. MobileFirst Platform (WAS single node) template nodes and components (continued).

Node	Components
MobileFirst Platform DB	**Database Server** DB2 database server installation. **MFP Administration DB** MobileFirst administration database schema installation. **MFP Runtime DB** MobileFirst runtime database schema installation. **Default add disk** Disk size configuration.

MobileFirst Platform (WAS server farm) template

Figure 9-5 shows the composition of the "MobileFirst Platform (WAS server farm)" template.

Figure 9-5. MobileFirst Platform (WAS server farm) template

The "MobileFirst Platform (WAS server farm)" template is composed of the following nodes and components:

Table 9-36. MobileFirst Platform (WAS server farm) template nodes and components.

Node	Components
IHS Server	**IBM HTTP servers** IBM HTTP Server installation. **MFP IHS Configuration** Automatic configuration of IBM HTTP Server.

Table 9-36. MobileFirst Platform (WAS server farm) template nodes and components (continued).

Node	Components
MobileFirst Platform Server	**Standalone server** WebSphere Application Server full profile server installation. **Restriction:** Do not change the values for the following component attributes: • Cell name • Node name • Profile name If you change any of these attributes, the deployment of patterns that are based on this template fails. **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Administration** MobileFirst Server Administration web application including MobileFirst Operations Console. **MFP Server Runtime Deployment** Runtime context root configuration. **MobileFirst App** MobileFirst application to be added to the deployment. **MobileFirst Adapter** MobileFirst adapter to be added to the deployment. **MFP Server Application Adapter Deployment** Application and adapter deployment to the MobileFirst Server. **Base Scaling Policy** VM scaling policy: number of VMs.
MobileFirst Platform DB	**Database Server** DB2 database server installation. **MFP Administration DB** MobileFirst administration database schema installation. **MFP Runtime DB** MobileFirst runtime database schema installation. **Default add disk** Disk size configuration.

MobileFirst Platform (WAS ND) template

Figure 9-6 shows the composition of the "MobileFirst Platform (WAS ND)" template.

Figure 9-6. MobileFirst Platform (WAS ND) template

The "MobileFirst Platform (WAS ND)" template is composed of the following nodes and components:

Table 9-37. MobileFirst Platform (WAS ND) template nodes and components.

Node	Components
IHS Server	**IBM HTTP servers** IBM HTTP Server installation. **MFP IHS Configuration** Automatic configuration of IBM HTTP Server.

Table 9-37. MobileFirst Platform (WAS ND) template nodes and components (continued).

Node	Components
DmgrNode	**Deployment manager** WebSphere Application Server deployment manager installation. **Restriction:** Do not change the values for the following component attributes: • Cell name • Node name • Profile name If you change any of these attributes, the deployment of patterns that are based on this template fails. **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Administration** MobileFirst Server Administration web application including MobileFirst Operations Console. **MFP Runtime** Runtime WAR file. **MFP Server Runtime Deployment** Runtime context root configuration. **MFP Application** MobileFirst application to be added to the deployment. **MFP Adapter** MobileFirst adapter to be added to the deployment. **MFP Server Application Adapter Deployment** Application and adapter deployment to the MobileFirst Server.
MobileFirst Platform DB	**Database Server** DB2 database server installation. **MFP Administration DB** MobileFirst administration database schema installation. **MFP Runtime DB** MobileFirst runtime database schema installation. **Default add disk** Disk size configuration.

Table 9-37. MobileFirst Platform (WAS ND) template nodes and components (continued).

Node	Components
CustomNode	**Custom nodes** Details of the cells and nodes in the clusters of WebSphere Application Server Network Deployment servers. **Restriction:** Do not change the values for the following component attributes: • Cell name • Node name • Profile name If you change any of these attributes, the deployment of patterns that are based on this template fails. **MFP Open Firewall Ports for WAS** Ports that must be open to enable connection to the database server and the LDAP server. **Base scaling policy** Number of virtual machine instances required for the chosen topology.

MobileFirst Platform Application Center (Liberty single node) template

Figure 9-7 shows the composition of the "MobileFirst Platform Application Center (Liberty single node)" template.

Figure 9-7. MobileFirst Platform Application Center (Liberty single node) template

The "MobileFirst Platform Application Center (Liberty single node)" template is composed of the following nodes and components:

Table 9-38. MobileFirst Platform Application Center (Liberty single node) template nodes and components.

Node	Components
MFP AppCenter DB	**Database Server** DB2 database server installation. **Default add disk** Disk size configuration.

Table 9-38. MobileFirst Platform Application Center (Liberty single node) template nodes and components (continued).

Node	Components
MFP AppCenter Server	**Liberty profile server** WebSphere Application Server Liberty profile server installation. **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Application Center** This script package sets up the MobileFirst Application Center server in a WebSphere Application Server full profile or WebSphere Application Server Liberty profile server.

MobileFirst Platform Application Center (WAS single node) template

Figure 9-8 shows the composition of the "MobileFirst Platform Application Center (WAS single node)" template.

Figure 9-8. MobileFirst Platform Application Center (WAS single node) template

The "MobileFirst Platform Application Center (WAS single node)" template is composed of the following nodes and components:

Table 9-39. MobileFirst Platform Application Center (WAS single node) template nodes and components.

Node	Components
MFP AppCenter DB	**Database Server** DB2 database server installation. **Default add disk** Disk size configuration.

Table 9-39. MobileFirst Platform Application Center (WAS single node) template nodes and components (continued).

Node	Components
MFP AppCenter Server	**Standalone server** WebSphere Application Server full profile server installation. **Restriction:** Do not change the values for the following component attributes: • Cell name • Node name • Profile name If you change any of these attributes, the deployment of patterns that are based on this template fails. **MFP WAS SDK Level** Purpose of this script is to set the required SDK level as the default SDK for the WAS Profile **MFP Server Prerequisite** Prerequisites for MobileFirst Server installation including SSL and Ant. **MFP Server Application Center** This script package sets up the MobileFirst Application Center server in a WebSphere Application Server full profile or WebSphere Application Server Liberty profile server.

Script packages for MobileFirst Server

IBM MobileFirst Platform Foundation System Pattern provides script packages that are the building blocks to compose various pattern topologies.

The following sections list and describe the parameters for each script package.
- "MFP Administration DB" on page 9-109
- "MFP Analytics" on page 9-109
- "MFP IHS Configuration" on page 9-112
- "MFP Open Firewall Ports for WAS" on page 9-113
- "MFP WAS SDK Level" on page 9-114
- "MFP Runtime DB" on page 9-115
- "MFP Server Administration" on page 9-115
- "MFP Server Application Adapter Deployment" on page 9-119
- "MFP Server Application Center" on page 9-120
- MFP Server Prerequisite
- MFP Server Runtime Deployment

MFP Administration DB

This script package sets up the administration database schema in a DB2 database. It must be used with the Database Server (DB2) software component.

Table 9-40. MFP Administration DB.

Parameter	Description
db_user	Mandatory. User name to create the Administration database. It can be mapped to the Instance name of the Database Server component. Default value: `db2inst1`.
db_name	Mandatory. Database name to create the Administration database. Default value: `WLADM`.
db_password	Mandatory. User password to create the Administration database. It can be mapped to the Instance owner password of the Database Server component. Default value: `passw0rd` (as pattern level parameter).
other_db_args	Mandatory. Four parameters to create the Administration database:**SQL type**, **Codeset**,**Territory** and **Collate**. Default value: `DB2 UTF-8 US SYSTEM`.

MFP Analytics

This script package sets up the MobileFirst Analytics server in a WebSphere Application Server full profile or WebSphere Application Server Liberty profile server, and sets up the connection and mapping of Analytics administration security roles to an external TDS or AD server. It must be used with the WebSphere Application Server Liberty profile server or WebSphere Application Server full profile (display name: Standalone server) software component . It must be installed after the Liberty profile or Standalone server software component.

Table 9-41. MFP Analytics.

Parameter	Description
WAS_ROOT	Mandatory. • If Analytics is installed on WebSphere Application Server Liberty profile, specify the installation directory of the WebSphere Application Server Liberty profile for Analytics. • If Analytics is installed on WebSphere Application Server full profile, specify the installation directory of the WebSphere Application Server full profile for Analytics.

Table 9-41. MFP Analytics (continued).

Parameter	**Description**
HEAP_MIN_SIZE	WebSphere Application Server full profile only. Depending on the amount of Analytics data that is generated, more memory is required for more data handling. Set this to allow larger minimum heap size for WebSphere Application Server full profile. Make sure the memory size specified in the Core OS component of MobileFirst Analytics is larger than this. It is recommended to set the same value as **`HEAP_MAX_SIZE`**. Default value: `4096` (MB).
`HEAP_MAX_SIZE`	WebSphere Application Server full profile only. Depending on the amount of Analytics data that is generated, more memory is required for more data handling. Set this to allow larger maximum heap size for WebSphere Application Server full profile. Make sure the memory size specified in the Core OS component of MobileFirst Analytics is larger than this. It is recommended to set the same value as**`HEAP_MIN_SIZE`**. Default value: `4096` (MB).
`WAS_admin_user`	WebSphere Application Server full profile only. WebSphere Application Server full profile admin user for the Analytics server. For WebSphere Application Server Liberty profile, leave the default value unchanged.
`WAS_admin_password`	WebSphere Application Server full profile only. WebSphere Application Server full profile admin user password for the Analytics server. For WebSphere Application Server Liberty profile, leave the default value unchanged.
`admin_user`	Mandatory. • If LDAP repository not enabled, create a default administration user for MobileFirst Analytics console protection. • If LDAP repository is enabled, specify the user name that has MobileFirst Analytics administration privilege. The value is stored in the LDAP repository.

Table 9-41. MFP Analytics (continued).

Parameter	Description
admin_password	Mandatory. • If an LDAP repository is not enabled, specify the password for the default administration user for MobileFirst Analytics console protection. • If an LDAP repository is enabled, specify the admin user password. The value is stored in the LDAP repository.
LDAP_TYPE	(LDAP parameter) Mandatory. LDAP server type of your user registry: **None** LDAP connection is disabled. When this is set, all the other LDAP parameters are treated as placeholders only. **TivoliDirectoryServer** Select this if the LDAP repository is an IBM Tivoli Directory Server. **ActiveDirectory** Select this if the LDAP repository is a Microsoft Active Directory. Default value: `None`.
LDAP_IP	(LDAP parameter). LDAP server IP address.
LDAP_SSL_PORT	(LDAP parameter) LDAP port for secure connection.
LDAP_PORT	(LDAP parameter) LDAP port for non-secure connection.
BASE_DN	(LDAP parameter) Base DN.
BIND_DN	(LDAP parameter) Bind DN.
BIND_PASSWORD	(LDAP parameter) Bind DN password.
REQUIRE_SSL	(LDAP parameter) Set it to `true` for secure connection to LDAP server. • When it is `true`, **LDAP_SSL_PORT** is used and **CERT_FILE_PATH** is required to locate the certification file of the LDAP server. • When it is `false`, **LDAP_PORT** is used. Default value: `false`.
USER_FILTER	(LDAP parameter) LDAP user filter that searches the existing user registry for users.
GROUP_FILTER	(LDAP parameter) LDAP group filter that searches the existing user registry for groups.
LDAP_REPOSITORY_NAME	(LDAP parameter) LDAP server name.
CERT_FILE_PATH	(LDAP parameter) Target path of the uploaded LDAP server certification. It is mandatory when **REQUIRE_SSL** is set to `true`.

Table 9-41. MFP Analytics (continued).

Parameter	Description
mfpadmin	(LDAP parameter) Admin role for MobileFirst Server: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: None.
mfpdeployer	(LDAP parameter) Deployer role for MobileFirst Server: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: None.
mfpmonitor	(LDAP parameter) Monitor role for MobileFirst Server: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: None.
mfpoperator	(LDAP parameter) Operator role for MobileFirst Server: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: None.

MFP IHS Configuration

This script package configures the IBM HTTP Server to work as a load balancer for multiple instances of MobileFirst Server. It must be used with the IBM HTTP servers software component . It must be installed after the IBM HTTP servers software component.

Table 9-42. MFP IHS Configuration.

Parameter	Description
WAS_ROOT	Mandatory. Installation directory of WebSphere Application Server Liberty profile or WebSphere Application Server full profile in the MobileFirst Platform Server node, or installation directory of Deployment manager in the DmgrNode node. In the pattern templates, it is mapped to output attribute**install_directory** of Liberty profile server, Standalone server, or Deployment manager.
profile_name	Optional. The profile name that contains the files for the WebSphere Application Server runtime environment. In the pattern templates, it is mapped to output attribute **dmgr_profile_name** of Deployment manager or **sa_profile_name** of Standalone server.
runtime_contextRoot_list	Mandatory. Runtime context root list that allows IHS to route requests that have matching context roots. Use semicolons (;) to separate the runtime context roots. For example, `HelloMobileFirst;HelloWorld` **Important:** It must align with the context root specified in the MFP Server Runtime Deployment. Otherwise, IHS cannot correctly route requests that contain the Runtime context root.
http_port	Mandatory. Open the firewall port in the IHS Server node to allow the HTTP transport from IHS Server to MobileFirst Server. Must be `9080`.
https_port	Mandatory. Open the firewall port in the IHS Server node to allow the HTTPS transport from IHS Server to MobileFirst Server. Must be `9443`.
server_hostname	Mandatory. Host name of IBM HTTP servers. It is mapped to the **host** output attribute of IBM HTTP servers in the pattern template.

MFP Open Firewall Ports for WAS

This script package is only applicable for Custom nodes in the MobileFirst (WAS ND) pattern template (WebSphere Application Server Network Deployment). Its purpose is to open the necessary firewall ports of the Custom nodes that host the MobileFirst Administration Services and runtime. As well as defining some WebSphere Application Server predefined ports, you need to specify the other ports for connecting to the DB2 server and the LDAP server.

Table 9-43. MFP Open Firewalls for WAS.

Parameter	Description
WAS_ROOT	Mandatory. Installation directory of WebSphere Application Server Network Deployment Custom nodes in the CustomNode node. In the pattern templates, it is mapped to output attribute **install_directory** of Custom nodes server.
profile_name	Mandatory. The profile name that contains the files for the WebSphere Application Server runtime environment. In the pattern templates, it is mapped to output attribute **cn_profile_name** of Custom nodes.
WAS_admin_user	Mandatory. It is mapped to the **was_admin** output attribute of Custom nodes in the pattern template.
Ports	Mandatory. Other ports that need to be opened for connecting to DB2 server and LDAP server (optional). Port values can be separated by semicolons; for example, '50000;636' Default value: 50000.

MFP WAS SDK Level

This script package is only applicable where ever the WAS Profiles are available in the pattern template (WebSphere Application Server Network Deployment).

Table 9-44. MFP WAS SDK Level. The purpose of this script is to set the required SDK level as the default SDK for the WAS Profile.

Parameter	Description
WAS_ROOT	Installation directory of WebSphere Application Server Liberty profile or WebSphere Application Server full profile in the MobileFirst Platform Server node or the installation directory of the Deployment manager in the DmgrNode node. In the pattern templates, it is mapped to output attribute**install_directory** of Liberty profile server, Standalone server, or Deployment manager.
profile_name	The profile name that contains the files for the WebSphere Application Server runtime environment. In the pattern templates, it is mapped to output attribute **dmgr_profile_name** of Deployment manager or **sa_profile_name** of Standalone server.
SDK_name	Name of the SDK that needs to be enabled for this WebSphere installation

MFP Runtime DB

This script package sets up the runtime database schema in a DB2 database.

Table 9-45. MFP Runtime DB. This script package sets up the runtime database schema in a DB2 database. It must be used with the Database Server (DB2) software component.

Parameter	Description
`db_user`	Mandatory. User name to create the Runtime database. It can be mapped to the Instance name of the Database Server component. Default value: `db2inst1`.
`db_name`	Mandatory. Database name to create the Runtime database. Default value: `WLRTIME`.
`db_password`	Mandatory. User password to create the Runtime database. It can be mapped to the Instance owner password of the Database Server component. Default value: `passw0rd` (as pattern level parameter).
`other_db_args`	Mandatory. Four parameters to create the Runtime database:**`SQL type`**, **`Codeset`**,**`Territory`** and **`Collate`**. Default value: `DB2 UTF-8 US SYSTEM`.

MFP Server Administration

This script package sets up the MobileFirst Administration component (including the MobileFirst Operations Console) in a WebSphere Application Server full profile or WebSphere Application Server Liberty profile server, and setting up the connection and mapping administration security roles to an external TDS or AD server.

The script package must be used with the WebSphere Application Server Liberty profile server software component or the WebSphere Application Server full profile software component (display name: Standalone server), and must be installed after the MFP Server Prerequisite but prior to any other MFP * Script Packages in the MobileFirst Platform Server VM node.

Table 9-46. MFP Server Administration.

Parameter	Description
`WAS_ROOT`	Mandatory. Installation directory of WebSphere Application Server Liberty profile or WebSphere Application Server full profile in the MobileFirst Platform Server node or the installation directory of the Deployment manager in the DmgrNode node. In the pattern templates, it is mapped to output attribute**`install_directory`** of Liberty profile server, Standalone server, or Deployment manager.
`profile_name`	Optional. The profile name that contains the files for the WebSphere Application Server runtime environment. In the pattern templates, it is mapped to output attribute **`dmgr_profile_name`** of Deployment manager or **`sa_profile_name`** of Standalone server.

Table 9-46. MFP Server Administration (continued).

Parameter	Description
NUMBER_OF_CLUSTERMEMBERS	Optional. Only applicable for the MobileFirst Platform (WAS ND) pattern template. It specifies the number of cluster members for the cluster to deploy the MFP administration service. Default value: 2.
db_user	Mandatory. User name that created the Administration database. It is mapped to the **db_user** output attribute of the MFP Administration DB script package in the pattern template.
db_name	Mandatory. Name of the Administration database. It is mapped to the**db_name** output attribute of the MFP Administration DB script package in the pattern template.
db_password	Mandatory. password for user who created the Administration database. It is mapped to the **db_password** output attribute of the MFP Administration DB script package in the pattern template.
db_ip	IP address of the DB server where the Administration database is installed. It is mapped to the **IP output** attribute of the Database Server software component in the pattern template.
db_port	Port number of the DB server where the Administration database is installed. It is mapped to the **instancePort** output attribute of the Database Server software component in the pattern template.
admin_user	User name that has MobileFirst Server administration privilege. • When **LDAP_TYPE** is None, create the default admin user. • When **LDAP_TYPE** is set to TivoliDirectoryServer or ActiveDirectory and other LDAP parameters are specified according to your LDAP server configuration, the **admin_user** value should be taken from the configured LDAP user repository. Not required when the MobileFirst Server is to be deployed on a single node of WebSphere Application Server full profile.
admin_password	Password of the admin user. • When **LDAP_TYPE** is None, create the default admin user password. • When an external LDAP server is configured, the user password is taken from the LDAP repository. Not required when the MobileFirst Server is to be deployed on a single node of WebSphere Application Server full profile.

Table 9-46. MFP Server Administration (continued).

Parameter	Description
install_console	Whether the MobileFirst Operations Console is to be deployed in the MobileFirst Platform Server node. Default value: Selected. (Check box)
WAS_admin_user	Optional. When the MobileFirst Server is deployed on WebSphere Application Server full profile, it is mapped to the **was_admin**output attribute of Standalone server in the pattern template. When the MobileFirst Server is deployed on WebSphere Application Server Network Deployment, it is mapped to the **was_admin** output attribute of Deployment manager in the pattern template.
WAS_admin_password	Optional. When the MobileFirst Server is deployed on WebSphere Application Server full profile, it is mapped to the**was_admin_password** output attribute of Standalone server in the pattern template. When the MobileFirst Server is deployed on WebSphere Application Server Network Deployment, it is mapped to the **was_admin_password** output attribute of Deployment manager in the pattern template.
server_hostname	Mandatory. Host name of the MobileFirst Server or Deployment manager. Mapped to the **host output** attribute of Liberty profile server, Standalone Server, or Deployment manager.
server_farm_mode	Mandatory. Whether the MobileFirst Server is to be deployed in server farm mode. Must be selected for a server farm topology and must be cleared for a standalone topology. Default value: set according to the topology defined in the pattern template.
webserver_ip	Optional. When IBM HTTP servers is deployed in the pattern template, this parameter is mapped to the **IP** output attribute of IBM HTTP servers.

Table 9-46. MFP Server Administration (continued).

Parameter	Description
LDAP_TYPE	(LDAP parameter) Mandatory. LDAP server type of your user registry. One of the following values: • None – LDAP connection is disabled. When this value is selected, all the other LDAP parameters are treated as placeholders only. • TivoliDirectoryServer: Select this value if the LDAP repository is IBM Tivoli Directory Server • ActiveDirectory: Select this value if the LDAP repository is Microsoft Active Directory Default value: None.
LDAP_IP	(LDAP parameter) LDAP server IP address.
LDAP_SSL_PORT	(LDAP parameter) LDAP port for secure connection.
LDAP_PORT	(LDAP parameter) LDAP port for non-secure connection.
BASE_DN	(LDAP parameter) Base DN.
BIND_DN	(LDAP parameter) Bind DN.
BIND_PASSWORD	(LDAP parameter) Bind DN password.
REQUIRE_SSL	(LDAP parameter) Set to true for secure connection to LDAP server. • When true, the**LDAP_SSL_PORT** is used and **CERT_FILE_PATH** is required to locate the certification file of the LDAP server. • When false, **LDAP_PORT** is used. Default value: false.
USER_FILTER	(LDAP parameter) User filter that searches the existing user registry for users.
GROUP_FILTER	(LDAP parameter) LDAP group filter that searches the existing user registry for groups.
LDAP_REPOSITORY_NAME	(LDAP parameter) LDAP server name.
CERT_FILE_PATH	(LDAP parameter) Target path of the uploaded LDAP server certification. It is mandatory when **REQUIRE_SSL** is set to true.
mfpadmin	Admin role for MobileFirst Server. One of the following values: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: None.

Table 9-46. MFP Server Administration (continued).

Parameter	Description
mfpdeployer	Deployer role for MobileFirst Server. One of the following values: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: `None`.
mfpmonitor	Monitor role for MobileFirst Server. One of the following values: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: `None`.
mfpoperator	Operator role for MobileFirst Server. One of the following values: **None** No user. **AllAuthenticatedUsers** Authenticated users **Everyone** All users. Default value: `None`.

MFP Server Application Adapter Deployment

This script package deploys applications and adapters to the MobileFirst Server. It must be installed after the corresponding MFP Server Runtime Deployment script package that installed the runtime where the application and adapter are to be deployed.

Table 9-47. MFP Server Application Adapter Deployment.

Parameter	Description
artifact_dir	Mandatory. Installation path of application and adapter for deployment. It is mapped to the **target_path**output attribute of the MobileFirst App component in the pattern template.
admin_context	Mandatory. Must be `mfpadmin`.

Table 9-47. MFP Server Application Adapter Deployment (continued).

Parameter	Description
runtime_context	Mandatory. Align with the runtime context root specified in the MFP Server Runtime Deployment component. It is mapped to **runtime_contextRoot** output attribute of the MFP Server Runtime Deployment component.
deployer_user	Mandatory. User account with application and adapter deployment privilege. Set as pattern level parameter in the pattern template.
deployer_password	Mandatory. User password with application and adapter deployment privilege. Set as pattern level parameter in the pattern template.
webserver_ip	Optional. When IBM HTTP servers is deployed in the pattern template, it is mapped to the same output attribute of MFP Server Administration.

MFP Server Application Center

This script package sets up the MobileFirst Application Center server in a WebSphere Application Server full profile or WebSphere Application Server Liberty profile server. It must be used with the WebSphere Application Server Liberty profile server and MFP Server Prerequisite or WebSphere Application Server full profile (**Standalone server**), MFP WAS SDK Level and MFP Server Prerequisite. It must be installed after the Liberty profile or Standalone server software component.

Table 9-48. MFP Server Application Center.

Parameter	Description
WAS_ROOT	Mandatory. Installation directory of WebSphere Application Server Liberty profile or WebSphere Application Server full profile in the MobileFirst Platform Server node. In the pattern templates, it is mapped to output attribute**install_directory** of Liberty profile server or Standalone server.
profile_name	The profile name that contains the files for the WebSphere Application Server runtime environment. In the pattern templates, it is mapped to output attribute **sa_profile_name** of Standalone server.
db_instance	Name of the database instance. It is mapped to the **instancePort** output attribute of the **Database Server** software component in the pattern template.

Table 9-48. MFP Server Application Center (continued).

Parameter	Description
db_user	User name that created the Administration database. It is mapped to the **db_user** output attribute of the MFP Administration DB script package in the pattern template.
db_name	Name of the Administration database. It is mapped to the**db_name** output attribute of the MFP Administration DB script package in the pattern template.
db_password	Password for user who created the Administration database. It is mapped to the **db_password** output attribute of the MFP Administration DB script package in the pattern template.
db_ip	IP address of the DB server where the Administration database is installed. It is mapped to the **IP output** attribute of the Database Server software component in the pattern template.
db_port	Port number of the DB server where the Administration database is installed. It is mapped to the **instancePort** output attribute of the Database Server software component in the pattern template.
admin_user	User name that has MobileFirst Server administration privilege. In the pattern template, it is associated with the parameter of the same name in the MFP Server Administration script package as a pattern level parameter to ensure they are set to the same value
admin_password	admin user password. In the pattern template, it is associated with the parameter of the same name in the MFP Server Administration script package as a pattern level parameter to ensure they are set to the same value
WAS_admin_user	Mandatory for WebSphere Application Server. Optional for WebSphere Application Server Liberty. When the MobileFirst Server is deployed on WebSphere Application Server full profile, it is mapped to the **was_admin**output attribute of Standalone server in the pattern template. When the MobileFirst Server is deployed on WebSphere Application Server Network Deployment, it is mapped to the **was_admin** output attribute of Deployment manager in the pattern template.

Table 9-48. MFP Server Application Center (continued).

Parameter	Description
WAS_admin_password	Mandatory for WebSphere Application Server. Optional for WebSphere Application Server Liberty. When the MobileFirst Server is deployed on WebSphere Application Server full profile, it is mapped to the**was_admin_password** output attribute of Standalone server in the pattern template. When the MobileFirst Server is deployed on WebSphere Application Server Network Deployment, it is mapped to the **was_admin_password** output attribute of Deployment manager in the pattern template.
server_hostname	Host name of the MobileFirst Server. It is mapped to the **host** output attribute of Liberty profile server or Standalone Server.
LDAP_TYPE	LDAP server type of your user registry: **None** LDAP connection is disabled. When this is set, all the other LDAP parameters are treated as placeholders only. **TivoliDirectoryServer** Select this if the LDAP repository is an IBM Tivoli Directory Server. **ActiveDirectory** Select this if the LDAP repository is a Microsoft Active Directory. Default value: `None`.
LDAP_IP	(LDAP parameter). LDAP server IP address.
LDAP_SSL_PORT	(LDAP parameter) LDAP port for secure connection.
LDAP_PORT	(LDAP parameter) LDAP port for non-secure connection.
BASE_DN	(LDAP parameter) Base DN.
BIND_DN	(LDAP parameter) Bind DN.
BIND_PASSWORD	(LDAP parameter) Bind DN password.
REQUIRE_SSL	(LDAP parameter) Set it to `true` for secure connection to LDAP server. • When it is `true`, **LDAP_SSL_PORT** is used and **CERT_FILE_PATH** is required to locate the certification file of the LDAP server. • When it is `false`, **LDAP_PORT** is used. Default value: `false`.
USER_FILTER	(LDAP parameter) LDAP user filter that searches the existing user registry for users.

Table 9-48. MFP Server Application Center (continued).

Parameter	Description
GROUP_FILTER	(LDAP parameter) LDAP group filter that searches the existing user registry for groups.
LDAP_REPOSITORY_NAME	(LDAP parameter) LDAP server name.
CERT_FILE_PATH	(LDAP parameter) Target path of the uploaded LDAP server certification. It is mandatory when **REQUIRE_SSL** is set to `true`.
appcenteradmin	Admin role for MobileFirst Application Center. Use one of the following values: • None • No user • AllAuthenticatedUsers • Authenticated users • Everyone • All users • Default value: None

MFP Server prerequisite

This script package includes all prerequisites that are required to install the MobileFirst Server, including the DB2 JDBC driver and Apache Ant. The script package must be used with the WebSphere Application Server Liberty profile server software component or the WebSphere Application Server full profile software component (display name: Standalone server), and must be installed after the server software component but prior to any other MFP* script packages in the MobileFirst Platform Server node.

Table 9-49. MFP Server Prerequisite.

Parameter	Description
None	No parameters for this script package.

MFP Server Runtime Deployment

This script package installs the MobileFirst runtime in a WebSphere Application Server full profile or WebSphere Application Server Liberty profile server with the MobileFirst Operations Console installed. The script package also sets up the connection to the MobileFirst Analytics server. It must be installed after the MFP Server Administration script package.

Table 9-50. MFP Server Runtime Deployment.

Parameter	Description
WAS_ROOT	Mandatory. Installation directory of WebSphere Application Server Liberty profile or WebSphere Application Server full profile in the MobileFirst Platform Server node, or installation directory of Deployment manager in the DmgrNode node. In the pattern templates, it is mapped to output attribute **install_directory** of Liberty profile server or Standalone server.
profile_name	Optional. The profile name that contains the files for the WebSphere Application Server runtime environment. In the pattern templates, it is mapped to output attribute **dmgr_profile_name** of Deployment manager or **sa_profile_name** of Standalone server.
NUMBER_OF_CLUSTERMEMBERS	Optional. Only applicable for the MobileFirst Platform (WAS ND) pattern template. It specifies the number of cluster members for the cluster to deploy MFP runtime. Default value: 2.
db_ip	IP address of the DB server where the Runtime (and optional Reports) database is installed. It is mapped to the **IP output** attribute of the Database Server software component in the pattern template.
db_port	Port number of the DB server where the Runtime (and optional Reports) database is installed. It is mapped to the**instancePort** output attribute of the Database Server software component in the pattern template.
admin_user	Mandatory. User name that has MobileFirst Server administration privilege. In the pattern template, it is associated with the parameter of the same name in the MFP Server Administration script package as a pattern level parameter to ensure they are set to the same value
admin_password	Mandatory. admin user password. In the pattern template, it is associated with the parameter of the same name in the MFP Server Administration script package as a pattern level parameter to ensure they are set to the same value
runtime_path	Mandatory. Runtime WAR file installed path. For example: it can be mapped to the **target_path** output attribute of MFP Server Runtime in the pattern template.

Table 9-50. MFP Server Runtime Deployment (continued).

Parameter	Description
runtime_contextRoot	Mandatory. Runtime context root. Must start with a forward slash, /; for example, "/HelloWorld". It is set as a pattern level parameter in the pattern template.
rtdb_name	Mandatory. Name of the Runtime database. It is mapped to the**db_name** output attribute of the MFP Runtime DB script package in the pattern template.
rtdb_user	Mandatory. User that created the Runtime database. It is mapped to the **db_user** output attribute of the MFP Runtime DB script package in the pattern template.
rtdb_password	Mandatory. Password of the user that created the Runtime database. It is mapped to the **db_password** output attribute of the MFP Runtime DB script package in the pattern template.
rptdb_name	Optional. Name of the Reports database. It is mapped to the**db_name** output attribute of the MFP Reports DB script package in the pattern template. Leave blank if you do not want to connect to a Reports database.
rptdb_user	Optional. User that created the Reports database. It is mapped to the**db_user** output attribute of the MFP Reports DB script package in the pattern template.
rptdb_password	Optional. Password of the user that created the Reports database. It is mapped to the **db_password** output attribute of MFP Reports DB script package in the pattern template.
was_admin_user	Optional. When the MobileFirst Server is deployed on WebSphere Application Server full profile, it is mapped to the **was_admin**output attribute of Standalone server in the pattern template. When the MobileFirst Server is deployed on WebSphere Application Server Network Deployment, it is mapped to the **was_admin** output attribute of Deployment manager in the pattern template.

Table 9-50. MFP Server Runtime Deployment (continued).

Parameter	Description
was_admin_password	Optional. When the MobileFirst Server is deployed on WebSphere Application Server full profile, it is mapped to the**was_admin_password** output attribute of Standalone server in the pattern template. When the MobileFirst Server is deployed on WebSphere Application Server Network Deployment, it is mapped to the **was_admin_password** output attribute of Deployment manager in the pattern template.
server_farm_mode	Mandatory. Map it to the same attribute of MFP Server Administration.
server_hostname	Mandatory. Host name of the MobileFirst Server. It is mapped to the **host** output attribute of Liberty profile server, Standalone Server, or Deployment manager.
analytics_ip	Optional. MobileFirst Analytics Node IP address to enable the Analytics capability in the MFP Server Runtime.
analytics_admin_user	Optional. Administrator name of the MobileFirst Analytics server.
analytics_admin_password	Optional. Password of administrator of the MobileFirst Analytics server.

Upgrading IBM MobileFirst Platform Foundation System Pattern

To upgrade IBM MobileFirst Platform Application Pattern, upload the `.tgz` file that contains the latest updates.

Procedure

1. Log into IBM PureApplication System with an account that is allowed to upload new system plugins.
2. From the IBM PureApplication System console, navigate to **Catalog** > **System Plug-ins**.
3. Upload the IBM MobileFirst Platform Application Pattern `.tgz` file that contains the updates.
4. Enable the plugins you have uploaded.
5. Redeploy the pattern.

Administering MobileFirst applications

Run and maintain MobileFirst applications in production.

IBM MobileFirst Platform Foundation 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:

- Registering and configuring mobile applications to MobileFirst Server.
- Deploying and configuring 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.

Administration roles

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.

For more information about MobileFirst administration roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

Administration tools

MobileFirst Operations Console is not the only way to administer MobileFirst applications. IBM MobileFirst Platform Foundation also provides other tools to incorporate administration operations into your build and deployment process.

A set of REST services is available to perform administration operations. For API reference documentation of these services, see “REST API for the MobileFirst Server administration service” on page 8-7.

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 `mfpadm` command line tool. See “Administering MobileFirst applications through Ant” on page 10-23 or “Administering MobileFirst applications through the command line” on page 10-47.

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.

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, deploy it to a testing environment, and then to a production environment.

Deploying or updating an adapter to a production environment

Review this check-list before you deploy or update an adapter to a production environment.

About this task

Adapters contain the server-side code of applications that are deployed on and serviced by IBM MobileFirst Platform Foundation. Read this checklist before you deploy or update an adapter to a production environment. For more information about creating and building adapters, see “Developing the server side of a MobileFirst application” on page 7-187.

Adapters can be uploaded, updated, or configured while a production server is running. After all the nodes of a server farm receive the new adapter or configuration, all incoming requests to the adapter use the new settings.

Procedure

1. If you update an existing adapter in a production environment, make sure that this adapter contains no incompatibilities or regressions with existing applications that are registered to a server.

   The same adapter can be used by multiple applications, or by multiple versions of the same application, that are already published to the store and used. Before you update the adapter in a production environment, run non-regression tests in a test server against the new adapter and copies of the apps that are built for the test server.

2. For Java adapters, if the adapter uses Java URLConnection with HTTPS, make sure that the back-end certificates are in the MobileFirst Server keystore.

   For more information, see "Using SSL in HTTP adapters" on page 7-222. For more information about using self-signed certificates, see "Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates."

   Note: If the application server is WebSphere Application Server Liberty, then the certificates must also be in the Liberty truststore.
3. Verify the server-side configuration of the adapter.

   For more information about adapter configuration, see "Configuring adapters" on page 7-227.
4. Use the **`mfpadm deploy adapter`** and **`mfpadm adapter set user-config`** commands to upload the adapter and its configuration.

   For more information about **`mfpadm`** for adapters, see "Commands for adapters" on page 10-57.

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. 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.
2. Import the back-end server certificate into the MobileFirst keystore.
3. Deploy the new the MobileFirst keystore. For more information, see "Configuring the MobileFirst Server keystore" on page 7-316.

Example

The `CN` name of the back-end certificate must match what is configured in the adapter-descriptor `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 keystore:

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

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

```
keytool -list -keystore mfp.keystore
```

7. Deploy the new the MobileFirst Server keystore.

Building an application for a test or production environment

To build an application for a test or production environment, you must configure it for its target server. To build an application for a production environment, additional steps apply.

About this task

Procedure

1. Make sure that the target server keystore is configured.

   For more information, see “Configuring the MobileFirst Server keystore” on page 7-316.

2. If you plan to distribute the app installable artifact, increment the app version.

3. Before you build your app, configure it for the target server.

   You define the URL and runtime name of the target server in the client properties file. You can also change the target server by using the IBM MobileFirst Platform Command Line Interface (CLI). To configure the app for a target server without registering the app to a running server, you can use the **mfpdev app config server <server URL>** and **mfpdev app config runtime <runtime_name>** commands. Alternatively, you can register the app to a running server by running the **mfpdev app register** command. Use the public URL of the server. This URL is used by the mobile app to connect to MobileFirst Server. For example, to configure the app for a target server mfp.mycompany.com with a runtime that has the default name mfp, run

mfpdev app config server https://mfp.mycompany.com
and
mfpdev app config runtime mfp.

4. Configure the secret keys and authorized servers for your application.
   a. If your app implements certificate pinning, use the certificate of your target server.

      For more information about certificate pinning, see "Certificate pinning" on page 7-185.
   b. If your iOS app uses App Transport Security (ATS), configure ATS for your target server.
   c. To configure secure Direct Update for an Apache Cordova application, see "Implementing secure Direct Update on the client side" on page 7-239.
   d. If you develop your app with Apache Cordova, configure the Cordova Content Security Policy (CSP).
5. If you plan to use Direct Update for an application that is developed with Apache Cordova, archive the versions of the Cordova plug-ins you used to build the app.

   Direct Update cannot be used to update native code. If you changed a native library or one of the build tools in a Cordova project and uploaded such a file to MobileFirst Server, the server detects the difference and does not send any updates for the client application. The changes in the native library might include a different Cordova version, a newer Cordova iOS plug-in, or even an **mfpdev** plug-in fix pack that is newer than the one that was used to build the original application.
6. Configure the app for production use.
   a. Consider disabling printing to the device log.
   b. If you plan to use IBM MobileFirst Analytics, verify that your app sends collected data to the MobileFirst Server. For more information, see "Sending analytics" on page 11-37.
   c. Consider disabling features of your app that call the setServerURL API, unless you plan to make a single build for multiple test servers.
7. If you build for a production server and plan to distribute the installable artifact, archive the app source code to be able to run non regression-tests for this app on a test server.

   For example, if you later update an adapter, you might run non-regression tests on already distributed apps that use this adapter. For more information, see "Deploying or updating an adapter to a production environment" on page 10-2.
8. Optional: Create the application-authenticity file for your application.

   You use the application-authenticity file after you register the application to the server to enable the application-authenticity security check.
   - For more information, see "Enabling the application-authenticity security check" on page 7-282.
   - For more information about configuring application authenticity, see "Configuring the application-authenticity security check" on page 7-284.
   - For more information about registering an application to a production server, see "Registering an application to a production environment" on page 10-6.

What to do next

Register and configure the application in the production server. For more information, see "Registering an application to a production environment."

Registering an application to a production environment

When you register an application to a production server, you upload its application descriptor, define its license type, and optionally activate application authenticity.

Before you begin

- Verify that MobileFirst Server keystore is configured and is not the default keystore. Do not use a server in production with a default keystore. The MobileFirst Server keystore defines the identity of MobileFirst Server instances, and is used to digitally sign OAuth tokens and Direct Update packages. You must configure the server's keystore with a secret key before you use it in production. For more information, see "Configuring the MobileFirst Server keystore" on page 7-316.
- Deploy the adapters used by the app. For more information, see "Deploying or updating an adapter to a production environment" on page 10-2.
- Build the application for your target server. For more information, see "Building an application for a test or production environment" on page 10-4.

About this task

When you register an application with a production server, you upload its application descriptor, define its license type, and optionally activate application authenticity. You might also define your update strategy if an older version of your app is already deployed. Read the following procedure to learn about important steps, and about ways to automate them with the **mfpadm** program.

Procedure

1. If your MobileFirst Server is configured for token licensing, make sure that you have enough available tokens on the License Key Server. For more information, see "Token license validation" on page 10-83 and "Planning for the use of token licensing" on page 6-150.

   Tip: You can set the token license type of your app before you register the first version of your app. For more information, see "Setting the application license information" on page 10-80.
2. Transfer the application descriptor from a test server to the production server.

   This operation registers your application to the production server and upload its configuration. For more information about transferring an application descriptor, see "Transferring server-side artifacts to a test or production server" on page 10-7.
3. Set the application license information For more information, see "Setting the application license information" on page 10-80.
4. Configure the application-authenticity security check. For more information about configuring the application-authenticity security check, see "Configuring the application-authenticity security check" on page 7-284.

Note: You need the application binary file to create the application-authenticity file. For more information, see "Enabling the application-authenticity security check" on page 7-282.

5. If your application uses push notification, upload the push notification certificates to the server. You can upload the push certificates for your application with MobileFirst Operations Console. The certificates are common to all versions of an application.

   Note: You might not be able to test the push notification for your app with production certificates before your app is published to the store.
6. Verify the following items before you publish the application to the store.
   a. Test any mobile application management feature that you plan to use, such as disabling remote applications or displaying of an administrator message. For more information, see "Mobile-application management" on page 10-15.
   b. In the case of an update, define the update strategy. For more information, see "Updating MobileFirst apps in production" on page 10-14.

Transferring server-side artifacts to a test or production server

You can transfer an application configuration from a one server to another by using command-line tools or a REST API.

About this task

The application descriptor file is a JSON file that contains the description and configuration of your application. When you run an app that connects to a MobileFirst Server instance, the app must be registered with that server and configured. After you define a configuration for your app, you can transfer the application descriptor to another server, for example to a test server or to a production server. After you transfer the application descriptor to the new server, the app is registered with the new server. Different procedures are available, depending on whether you develop mobile applications and have access to the code, or whether you administer servers and do not have access to the code of the mobile app.

Important: If you import an application that includes authenticity data, and if the application itself has been recompiled since the authenticity data was generated, you must refresh the authenticity data. For more information, see "Configuring the application-authenticity security check" on page 7-284.

Procedure

- If you have access to the code of the mobile app, use the **`mfpdev app pull`** and **`mfpdev app push`** commands.
- If you do not have access to the code of the mobile app, use the administration service.

Transferring an application configuration by using mfpdev

After you have developed an application, you can transfer it from your development environment to a test or production environment.

Before you begin

- You must have an existing MobileFirst app on your local computer. The app must be registered to a MobileFirst Server. For information about creating a server profile, run **mfpdev app register**, or the topic about registering your type of app in the Developing applications section of this documentation.

- You must have connectivity from your local computer to the server that your app is currently registered to and to the server that you want to transfer your app to.
- You must have a server profile on the local computer for both the original MobileFirst Server and the server that you want to transfer your app to. For information about creating a server profile, run **mfpdev server add**.
- You must have the IBM MobileFirst Platform Command Line Interface (CLI) installed. For more information, see "Installing the MobileFirst Platform CLI" on page 7-15.

About this task

You use the **mfpdev app pull** command to send a copy of the server-side configuration files for your app to your local computer. Then you use the **mfpdev app push** command to send it to another MobileFirst Server. The **mfpdev app push** command also registers the app on the specified server.

You can also use these commands to transfer a runtime configuration from one server to another.

The configuration information includes the contents of the application descriptor, which uniquely identifies the app to the server and other information that is specific to the app. The configuration files are provided as compressed files (.zip format). The .zip files are placed in the directory *appName*/mobilefirst and named as follows:

appID-*platform*-*version*-artifacts.zip

where *appID* is the application name, *platform* is one of android, ios, or windows, and *version* is the version level of your app. For Cordova apps, a separate .zip file is created for each target platform.

When you use the **mfpdev app push** command, the application's client properties file is modified to reflect the profile name and URL of the new MobileFirst Server.

Procedure

1. On your development computer, navigate to a directory that is the root directory of your app or one of its subdirectories.
2. Run the **mfpdev app pull** command. If you specify the command with no parameters, the app is pulled from the default MobileFirst Server. You can also specify a particular server and its administrator password. For example, for an Android application named myapp1:

   ```
   $ cd myapp1
   $ mfpdev app pull Server10 -password secretPassword!
   ```

   This command finds the configuration files for the current application on the MobileFirst Server whose server profile is named Server10. Then, it sends the compressed file myapp1-android-1.0.0-artifacts.zip, which contains these configuration files, to the local computer and places it in the directory myapp1/mobilefirst.
3. Run the **mfpdev app push** command. If you specify the command with no parameters, the app is pushed to the default MobileFirst Server. You can also

specify a particular server and its administrator password. For example, for the same application that was pushed in the previous step:

```
$ mfpdev app push Server12 -password secretPass234!
```

This command sends the file `myapp1-android-1.0.0-artifacts.zip` to the MobileFirst Server whose server profile is named Server12, that has the administrator password `secretPass234!` The client properties file `myapp1/app/src/main/assets/mfpclient.properties` is modified to reflect that the server that the app is registered to is Server12, with the server's URL.

Results

The app's server-side configuration files are present on the MobileFirst Server that you specified in the **`mfpdev app push`** command. The app is registered to this new server.

What to do next

Test the app or deploy it on the MobileFirst Server that you have transferred it to.

Transferring an application configuration with the administration service

As an administrator, you can transfer an application configuration from one server to another by using the administration service of MobileFirst Server. No access to the application code is required, but the client app must be built for the target server.

Before you begin

- Build the client app for your target server. For more information, see “Building an application for a test or production environment” on page 10-4.

About this task

You download the application descriptor from the server where the application is configured and you deploy it to the new server. You can see the application descriptor with MobileFirst Operations Console.

Procedure

1. Optional: Review the application descriptor in the server where the application server is configured.

   Open the MobileFirst Operations Console for that server, select your application version, and go to the tab **Configuration Files**.
2. Download the application descriptor from the server where the application is configured. You can download it by using the REST API or **`mfpadm`**.

   Note: You can also export an application or application version from the MobileFirst Operations Console. See “Exporting and importing applications and adapters from the MobileFirst Operations Console” on page 10-12.
   - To download the application descriptor with the REST API, use the “Application Descriptor (GET)” on page 8-37 REST API.

     The following URL returns the application descriptor for the application of app ID `my.test.application`, for the `ios` platform, and version `0.0.1`. The call is made to MobileFirst Development Server.

`http://localhost:9080/mfpadmin/management-apis/2.0/runtimes/mfp/applications/my.test.application/ios/0.0.1/descriptor`

For example, you can use such URL with a tool like **curl**.

```
curl -user admin:admin http://[...]/ios/0.0.1/descriptor > desc.json
```

Change the following elements of the URL according to your server configuration:

- `9080` is the HTTP port of MobileFirst Development Server.
- `mfpadmin` is the context root of the administration service. This context root is `mfpadmin` in MobileFirst Development Server.

For information about the REST API, see "REST API for the MobileFirst Server administration service" on page 8-7.

- Download the application descriptor by using **mfpadm**.

  The **mfpadm** program is installed when you run the MobileFirst Server installer. You start it from the *product_install_dir*/`shortcuts/` directory, where *product_install_dir* indicates the installation directory of MobileFirst Server.

  The following example creates a password file, which is required by the **mfpadm** command, then downloads the application descriptor for the application of app ID `my.test.application`, for the `ios` platform, and version `0.0.1`. The provided URL is the HTTPS URL of MobileFirst Development Server.

```
prompt> echo password=admin > password.txt
prompt> mfpadm --url https://localhost:9443/mfpadmin --secure false --user admin \
        --passwordfile password.txt \
        app version mfp my.test.application ios 0.0.1 get descriptor > desc.json
prompt> rm password.txt
```

  Change the following elements of the command line according to your server configuration:

  - `9443` is the HTTPS port of MobileFirst Development Server.
  - `mfpadmin` is the context root of the administration service. This context root is `mfpadmin` in MobileFirst Development Server.
  - `--secure false` indicates that the server's SSL certificate is accepted even if self-signed or if created for a different host name from the server's host name used in the URL.

  For more information about the **mfpadm** program, see "Administering MobileFirst applications through the command line" on page 10-47.

3. Upload the application descriptor to the new server to register the app or update its configuration.

   You can upload it by using the REST API or **mfpadm**.

   - To upload the application descriptor with the REST API, use the "Application (POST)" on page 8-43 REST API.

     The following URL uploads the application descriptor to the `mfp` runtime. You send a POST request, and the payload is the JSON application descriptor. The call in this example is made to server that runs on the local computer and that is configured with an HTTP port set to 9081.

     `http://localhost:9081/mfpadmin/management-apis/2.0/runtimes/mfp/applications/`

     For example, you can use such URL with a tool like **curl**.

```
curl -H "Content-Type: application/json" -X POST -d @desc.json -u admin:admin \
        http://localhost:9081/mfpadmin/management-apis/2.0/runtimes/mfp/applications/
```

- Upload the application descriptor by using **mfpadm**.

  The following example creates a password file, which is required by the **mfpadm** command, then uploads the application descriptor for the application of app ID `my.test.application`, for the `ios` platform, and version `0.0.1`. The provided URL is the HTTPS URL of a server that runs on the local computer but is configured with an HTTPS port set to 9444, and for a runtime named `mfp`.

```
prompt> echo password=admin > password.txt
prompt> mfpadm --url https://localhost:9444/mfpadmin --secure false --user admin \
        --passwordfile password.txt \
        deploy app mfp desc.json
prompt> rm password.txt
```

Transferring server-side artifacts by using the REST API

Whatever your role, you can export applications, adapters, and resources for back-up or reuse purposes by using the MobileFirst Server administration service. As an administrator or deployer, you can also deploy an export archive to a different server. No access to the application code is required, but the client app must be built for the target server.

Before you begin

- Build the client app for your target server. For more information, see "Building an application for a test or production environment" on page 10-4.

About this task

The export API retrieves the selected artifacts for a runtime as a `.zip` archive. Use the deployment API to reuse archived content.

Important: Carefully consider your use case:

- The export file includes the application authenticity data. That data is specific to the build of a mobile app. The mobile app includes the URL of the server and its runtime name. Therefore, if you want to use another server or another runtime, you must rebuild the app. Transferring only the exported app files would not work.
- Some artifacts might vary from one server to another. Push credentials are different depending on whether you work in a development or production environment.
- The application runtime configuration (that contains the active/disabled state and the log profiles) can be transferred in some cases but not all.
- Transferring web resources might not make sense in some cases, for example if you rebuild the app to use a new server.

Procedure

- To export all resources, or a selected subset of resources, for one adapter or for all adapters, use the "Export adapter resources (GET)" on page 8-103 or "Export adapters (GET)" on page 8-105 API.
- To export all the resources under a specific application environment (such as Android or iOS), that is all the versions and all the resources for the version for that environment, use the "Export application environment (GET)" on page 8-106 API.

- To export all the resources for a specific version of an application (for example, version 1.0 or 2.0 of an Android application), use the "Export application environment resources (GET)" on page 8-107 API.
- To export a specific application or all applications for a runtime, use the "Export applications (GET)" on page 8-110 or "Export application resources (GET)" on page 8-109 API.

  Note: Credentials for push notification are not exported among the application resources.
- To export the adapter content, descriptor, license configuration, content, user configuration, keystore, and web resources of an application, use the "Export resources (GET)" on page 8-111 API.
- To export all or selected resources for a runtime, use the "Export runtime resources (GET)" on page 8-113 API. For example, you can use this general **curl** command to retrieve all resources as a .zip file.

```
curl -X GET -u admin:admin -o exported.zip
    "http://localhost:9080/worklightadmin/management-apis/2.0/runtimes/mfp/export/all"
```

- To deploy an archive that contains such web application resources as adapter, application, license configuration, keystore, web resource, use the "Deploy (POST)" on page 8-76 API. For example, you can use this **curl** command to deploy an existing .zip file that contains artifacts.

```
curl -X POST -u admin:admin -F
    file=@/Users/john_doe/Downloads/export_applications_adf_ios_2.zip
    "http://localhost:9080/mfpadmin/management-apis/2.0/runtimes/mfp/deploy/multi"
```

- To deploy application authenticity data, use the "Deploy Application Authenticity Data (POST)" on page 8-80 API.
- To deploy the web resources of an application, use the "Deploy a web resource (POST)" on page 8-83 API.

Results

If you deploy an export archive to the same runtime, the application or version is not necessarily restored as it was exported. That is, the redeployment does not remove subsequent modifications. Rather, if some application resources are modified between export time and redeployment, only the resources that are included in the exported archive are redeployed in their original state. For example, if you export an application with no authenticity data, then you upload authenticity data, and then you import the initial archive, the authenticity data is not erased.

Exporting and importing applications and adapters from the MobileFirst Operations Console

From the console, under certain conditions, you can export an application or one of its versions, and later import it to a different runtime on the same server or a different server. You can also export and reimport adapters. Use this capability for reuse or back-up purposes.

Before you begin

Open the MobileFirst Operations Console. For more information, see "Opening the MobileFirst Operations Console" on page 7-12.

About this task

If you are granted the **`mfpadmin`** administrator role and the **`mfpdeployer`** deployer role, you can export one version or all versions of an application. The application or version is exported as a `.zip` compressed file, which saves the application ID, descriptors, authenticity data, and web resources. You can later import the archive to redeploy the application or version to another runtime on the same or on a different server.

Important: Carefully consider your use case:

- The export file includes the application authenticity data. That data is specific to the build of a mobile app. The mobile app includes the URL of the server and its runtime name. Therefore, if you want to use another server or another runtime, you must rebuild the app. Transferring only the exported app files would not work.
- Some artifacts might vary from one server to another. Push credentials are different depending on whether you work in a development or production environment.
- The application runtime configuration (that contains the active/disabled state and the log profiles) can be transferred in some cases but not all.
- Transferring web resources might not make sense in some cases, for example if you rebuild the app to use a new server.

You can also transfer application descriptors by using the REST API or the `mfpadm` tool. For more information, see "Transferring an application configuration with the administration service" on page 10-9.

Procedure

1. From the navigation sidebar, select an application or application version, or an adapter.
2. Select **Actions** > **Export Application** or **Export Version** or **Export Adapter**.

   You are prompted to save the `.zip` archive file that encapsulates the exported resources. The aspect of the dialog box depends on your browser and the target folder depends on your browser settings.
3. Save the archive file.

   The archive file name includes the name and version of the application or adapter, for example `export_applications_com.sample.zip`.
4. To reuse an existing export archive, select **Actions** > **Import Application** or **Import Version** or **Import version**, browse to the archive, and click **Deploy**.

   The main console frame displays the details of the imported application or adapter.

Results

If you import to the same runtime, the application or version is not necessarily restored as it was exported. That is, the redeployment at import time does not remove subsequent modifications. Rather, if some application resources are modified between export time and redeployment at import time, only the resources that are included in the exported archive are redeployed in their original state. For example, if you export an application with no authenticity data, then you upload authenticity data, and then you import the initial archive, the authenticity data is not erased.

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.

When you upgrade an app, you can deploy a new app version and leave the old version working, or deploy a new app version and block the old version. In the case of an app developed with Apache Cordova, you can also consider only updating the Web resources.

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 application. For more information, see "Building an application for a test or production environment" on page 10-4
3. Register the app to MobileFirst Server and configure it.
4. Submit the new `.apk`, `.ipa`, `.appx`, or `.xap` files to their respective app stores.
5. Wait for review and approval, and for the apps to become available.
6. Optional - send notification message to users of the old version, announcing the new version. See "Displaying an administrator message" on page 10-18 and "Defining administrator messages in multiple languages" on page 10-19.

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 an administrator message" on page 10-18 and "Defining administrator messages in multiple languages" on page 10-19.
2. Increment the app version number.
3. Build and test your application. For more information, see "Building an application for a test or production environment" on page 10-4
4. Register the app to MobileFirst Server and configure it.
5. Submit the new `.apk`, `.ipa`, `.appx`, or `.xap` files to their respective app stores.
6. Wait for review and approval, and for the apps to become available.
7. Copy links to the new app version.
8. Block the old version of the app in MobileFirst Operations Console, supplying a message and link to the new version. See "Remotely disabling application access to protected resources" on page 10-17.

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.

Direct Update (no native code changes)

Direct Update is a mandatory upgrade mechanism that is used to deploy fast fixes to a production app. When you redeploy an app to MobileFirst Server without changing its version, MobileFirst Server directly pushes the updated web resources to the device when the user connects to the server. It does not push updated native code. Things to keep in mind when you consider a Direct Update include:

- Direct Update does **not** update the app version. The app remains at the same version, but with a different set of web resources. The unchanged version number can introduce confusion if used for the wrong purpose
- Direct Update also does not go through the app store review process because it is technically not a new release. This should not be abused because vendors can become displeased if you deploy a whole new version of your app that bypasses their review. It is your responsibility to read each store's usage agreements and abide by them. Direct Update is best used to fix urgent issues that cannot wait for several days.
- Direct Update is considered a security mechanism, and therefore it is mandatory, not optional. When you initiate the Direct Update, all users **must** update their app to be able to use it.
- Direct Update does not work if an application is compiled (built) with a different version of IBM MobileFirst Platform Foundation than the one that was used for the initial deployment.

Administering MobileFirst applications through the MobileFirst Operations Console

You can administer MobileFirst applications through the MobileFirst Operations Console by locking apps or denying access, or by displaying notification messages.

You can start the console by entering one of the following URLs:

- Secure mode for production or test: `https://`*`hostname`*`:`*`secure_port`*`/mfpconsole`
- Development: `http://`*`server_name`*`:`*`port`*`/mfpconsole`

You must have a login and password that grant you authorization to access the MobileFirst Operations Console. For more information, see “Configuring user authentication for MobileFirst Server administration” on page 6-166.

You can use the MobileFirst Operations Console to manage your applications.

From the MobileFirst Operations Console, you can also access the Analytics console and control the collection of mobile data for analysis by the Analytics server. For more information, see “Enabling or disabling data collection from the MobileFirst Operations Console” on page 11-23.

Note: Deleting an application from MobileFirst Operations Console will remove all push subscriptions on that application as well.

Mobile-application management

The MobileFirst mobile-application-management capabilities provide MobileFirst Server operators and administrators with granular control over user and device access to their applications.

MobileFirst Server tracks all attempts to access your mobile infrastructure, and stores information about the application, the user, and the device on which the application is installed. The mapping between the application, the user, and the device, forms the basis for the server's mobile-application management capabilities.

Use IBM MobileFirst Platform Operations Console to monitor and manage access to your resources:

- Search for a user by name, and view information about the devices and applications that they are using to access your resources.
- Search for a device by its display name, and view the users that are associated with the device, and the registered MobileFirst applications that are used on this device.
- Block access to your resources from all instances of your applications on a specific device. This is useful when a device is lost or stolen.
- Block access to your resources only for a specific application on a specific device. For example, if an employee changes departments, you can block the employee's access for an application of the previous department, but allow the employee access from other applications on the same device.
- Unregister a device, and delete all the registration and monitoring data that was gathered for the device.

Access-blocking has the following characteristics:

- The blocking operation is reversible. You can remove the block by changing the device or application status in MobileFirst Operations Console.
- The block applies only to protected resources. A blocked client can still use the application to access an unprotected resource. See Unprotected resources.
- Access to adapter resources on MobileFirst Server is blocked immediately when you select this operation. However, this might not be the case for resources on an external server because the application might still have a valid access token that has not expired.

Device status

MobileFirst Server maintains status information for every device that accesses the server. The possible status values are `Active`, `Lost`, `Stolen`, `Expired`, and `Disabled`. The default device status is `Active`, which indicates that access from this device is not blocked. You can change the status to `Lost`, `Stolen`, or `Disabled` to block access to your application resources from the device. You can always restore the `Active` status to allow access again. See "Managing device access in MobileFirst Operations Console" on page 10-17.

The `Expired` status is a special status that is set by MobileFirst Server after a preconfigured inactivity duration elapses since the last time that the device connected to this server instance. This status is used for license tracking, and it does not affect the access rights of the device. When a device with an `Expired` status reconnects to the server, its status is restored to `Active`, and the device is granted access the server.

Device display name

MobileFirst Server identifies devices by a unique device ID, which is assigned by the MobileFirst client SDK. Setting a display name for a device allows you to search for the device by its display name. Application developers can use the setDeviceDisplayName method of the WLClient class to set the device display name. Java adapter developers (including security-check developers) can also set the device display name by using the setDeviceDisplayName method of the com.ibm.mfp.server.registration.external.model MobileDeviceData class.

Managing device access in MobileFirst Operations Console

To monitor and manage device access to your resources, select the **Devices** tab in the MobileFirst Operations Console dashboard.

Use the search field to search for a device by the user ID that is associated with the device, or by the display name of the device (if set). See "Device display name" on page 10-16. You can also search for part of the user ID or the device display name (at least three characters).
The search results display all the devices that match the specified user ID or device display name. For each device, you can see the device ID and display name, the device model, the operating system, and the list of users IDs that are associated with the device.

The **Device Status** column shows the status of the device. You can change the status of the device to `Lost`, `Stolen`, or `Disabled`, to block access from the device to protected resources. Changing the status back to `Active` restores the original access rights.

You can unregister a device by selecting **Unregister** in the **Actions** column. Unregistering a device deletes the registration data of all the MobileFirst applications that are installed on the device. In addition, the device display name, the lists of users that are associated with the device, and the public attributes that the application registered for this device are deleted.

Note: The **Unregister** action is not reversible. The next time that one of the MobileFirst applications on the device attempts to access the server, it will be registered again with a new device ID. When you select to register the device again, the device status is set to `Active`, and the device has access to protected resources, regardless of any previous blocks. Therefore, if you want to block a device, do not unregister it. Instead, change the device status to `Lost`, `Stolen`, or `Disabled`.

To view of all the applications that were accessed on a specific device, select the expand arrow icon next to the device ID in the devices table. Each row in the displayed applications table contains the name of the application, and the application's access status (whether access to protected resources is enabled for this application on this device). You can change the application's status to `Disabled` to block access from the application specifically on this device.

Remotely disabling application access to protected resources

Learn how to remotely disable an application and deny it access to protected resources due to phase-out policy or identified security issues.

About this task

Use MobileFirst Operations Console (the console) to disable user access to a specific version of an application on a specific mobile operating system, and provide a custom message to the user.

Procedure

1. Select your application version from the **Applications** section of the console's navigation sidebar, and then select the application **Management** tab.
2. Change the status to **Access Disabled**.

3. In the **URL of latest version** field, optionally provide a URL for a newer 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 13-27.
4. In the **Default notification message** field, add the custom notification message to display when the user attempts to access the application. The following sample message directs users to upgrade to the latest version:

   ```
   This version is no longer supported. Please upgrade to the latest version.
   ```
5. In the **Supported locales** section, you can optionally provide the notification message in other languages. See the detailed instructions in "Defining administrator messages in multiple languages" on page 10-19.
6. Select **Save** to apply your changes.

Results

When a user runs an application that was remotely disabled, a dialog window with your custom message is displayed. The message is displayed on any application interaction that requires access to a protected resource, or when the application tries to obtain an access token. If you provided a version-upgrade URL, the dialog has a **Get new version** button for upgrading to a newer version, in addition to the default **Close** button. If the user closes the dialog window without upgrading the version, they can continue to work with the parts of the application that do not require access to protected resources. However, any application interaction that requires access to a protected resource causes the dialog window to be displayed again, and the application is not granted access to the resource.

Note: For cross-platform applications, you can customize the default remote-disable behavior: provide an upgrade URL for your application, as outlined in Step 3, and set the **`showCloseOnRemoteDisableDenial`** attribute in your application's `initOptions.js` file to `false`. If the attribute is not defined, define it. When an application-upgrade URL is provided and the value of **`showCloseOnRemoteDisableDenial`** is `false`, the **Close** button is omitted from the remote-disable dialog window, leaving only the **Get new version** button. This forces the user to upgrade the application. When no upgrade URL is provided, the **`showCloseOnRemoteDisableDenial`** configuration has no effect, and a single **Close** button is displayed.

Displaying an administrator message

Define a notification message that is displayed when the application first accesses MobileFirst Server.

About this task

Follow the outlined procedure to configure the notification message. You can use this message to notify application users of temporary situations, such as a planned service downtime.

Procedure

1. Select your application version from the **Applications** section of the MobileFirst Operations Console navigation sidebar, and then select the application **Management** tab.
2. Change the status to **Active and Notifying**.

3. Add a custom startup message. The following sample message informs the user of planned maintenance work for the application:

   ```
   The server will be unavailable on Saturday between 4 AM to 6 PM due to planned maintenance.
   ```
4. In the **Supported locales** section, you can optionally provide the notification message in other languages. See the detailed instructions in "Defining administrator messages in multiple languages."
5. Select**Save** to apply your changes.

Results

The message is displayed when the application first uses MobileFirst Server to access a protected resource (see "OAuth resource protection" on page 7-271), or obtain an access token (see "Overview of the MobileFirst security framework" on page 7-265). If the application acquires an access token when it starts, the message is displayed at this stage. Otherwise, the message is displayed on the first request from the application to access a protected resource or obtain an access token. The message is displayed only once, for the first interaction.

Defining administrator messages in multiple languages

You can display the administrator messages that you define in IBM MobileFirst Platform Operations Console in multiple languages.

About this task

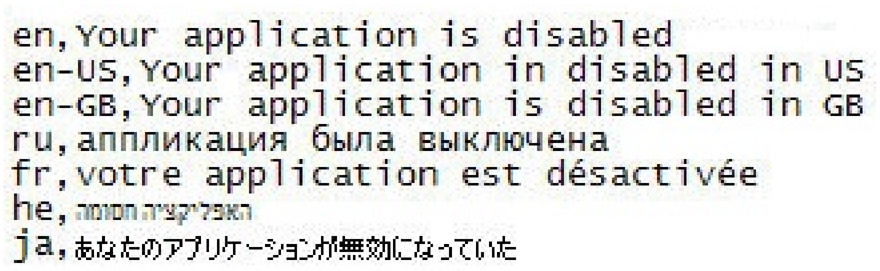
Follow the outlined procedure to configure multiple languages for displaying the application administration messages that you defined through the console (see "Remotely disabling application access to protected resources" on page 10-17 and "Displaying an administrator message" on page 10-18). The messages are sent based on the locale of the device, and must comply with the standards that the mobile operating system uses to specify locales.

Procedure

1. Select your application version from the **Applications** section of the MobileFirst Operations Console navigation sidebar, and then select the application **Management** tab.
2. Select the status **Active and Notifying** or **Access Disabled**.
3. Select **Update Locales**. In the **Upload File** section of the displayed dialog window, select **Upload**, and browse to the location of a CSV file that defines the locales.

   Each line in the CSV file contains a pair of comma-separated strings. The first string is the locale code (such as `fr-FR` for French (France) or `en` for English), and the second string is the message text in the corresponding language. The specified locale codes must comply with the standards that the mobile operating system uses to specify locales, such as ISO 639-1, ISO 3166-2, and ISO 15924.

   Note: To create the CSV file, you must use an editor that supports UTF-8 encoding, such as Notepad.
   Following is a sample CSV file that defines the same message for multiple locales:



```
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 10-1. Sample CSV file

4. In the **Verify notification message** section, you can see a table of the locale codes and messages from your CSV file. Verify the messages, and select **OK**. You can select **Edit**, at any time, to replace the locales CSV file. You can also use this option to upload an empty CSV file to remove all locales.
5. Select **Save** to apply your changes.

Results

The localized notification message is displayed on the user's mobile device, according to the locale of the device. If no message was configured for the device locale, the default message that you provided is displayed.

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 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.

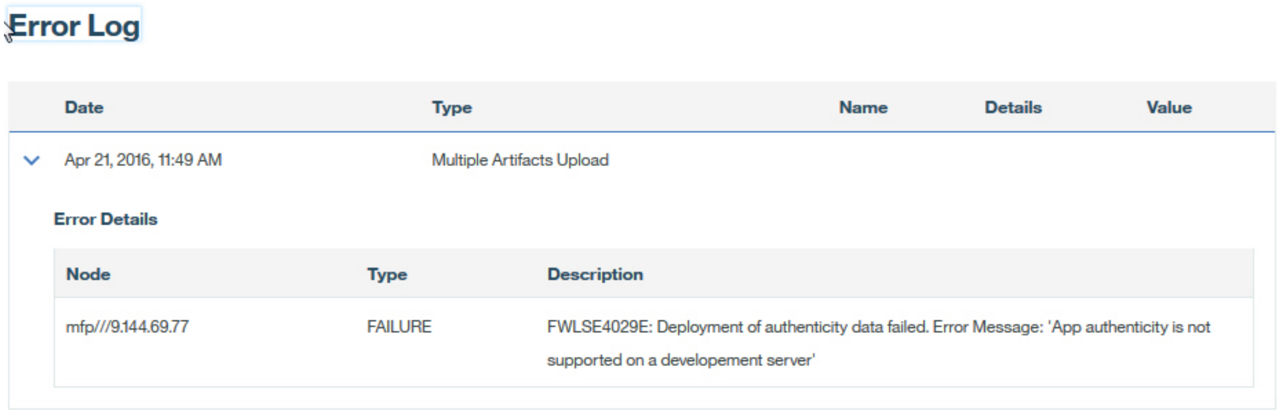
Error log of operations on runtime environments

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
Apr 21, 2016, 11:49 AM	Multiple Artifacts Upload			

Error Details

Node	Type	Description
mfp///9.144.69.77	FAILURE	FWLSE4029E: Deployment of authenticity data failed. Error Message: 'App authenticity is not supported on a developement server'

Figure 10-2. 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 **`mfp.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 10-3 on page 10-23.

Table 10-1. Fields in audit log records

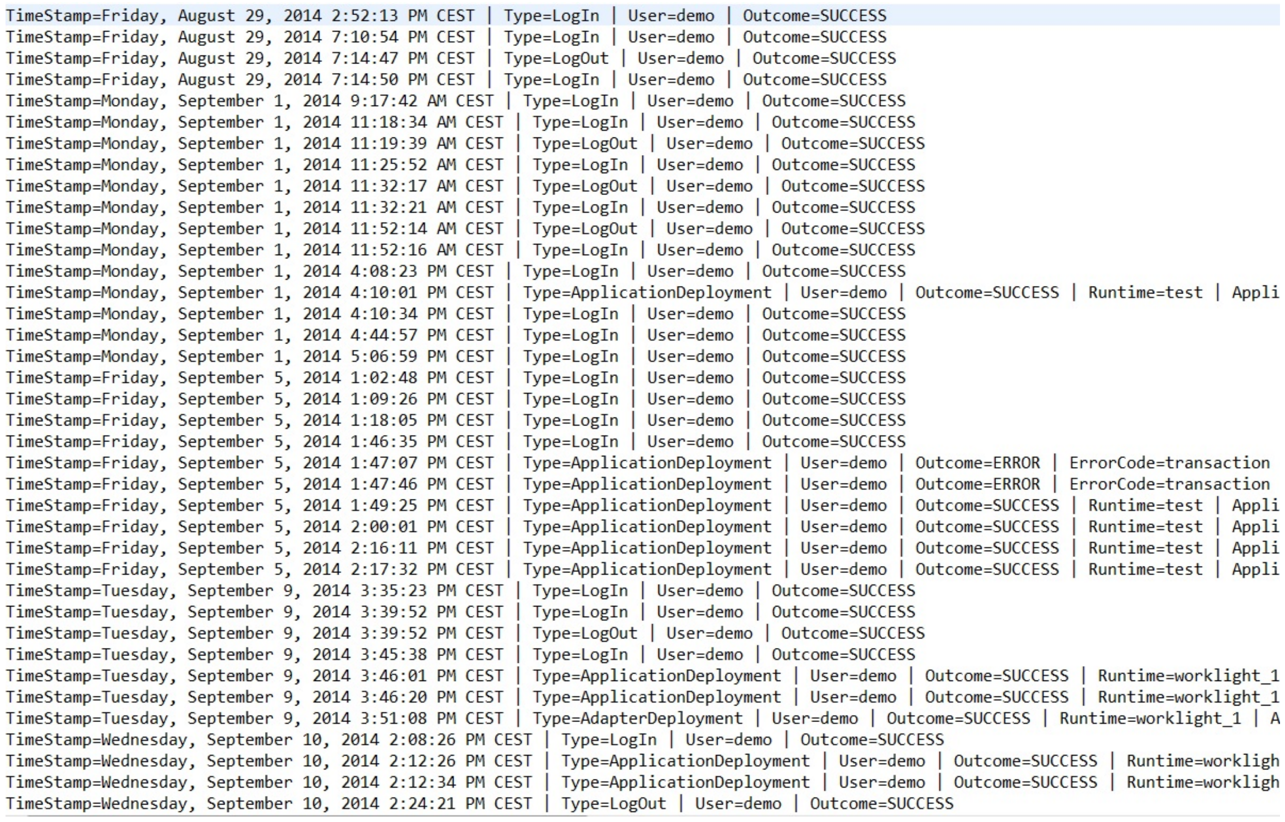
Field name	Description
`Timestamp`	Date and time when the record was created.
`Type`	The type of operation. See list of operation types for the possible values.
`User`	The `username` of the user who is signed in.
`Outcome`	The outcome of the operation; possible values are `SUCCESS`, `ERROR`, `PENDING`.
`ErrorCode`	If the outcome is `ERROR`, **`ErrorCode`** indicates what the error is.

Table 10-1. 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=workligh
TimeStamp=Wednesday, September 10, 2014 2:12:34 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=workligh
TimeStamp=Wednesday, September 10, 2014 2:24:21 PM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
```

Figure 10-3. Sample audit log of MobileFirst administration operations

Administering MobileFirst applications through Ant

You can administer MobileFirst applications through the **mfpadm** Ant task.

Comparison with other facilities

You can execute administration operations with IBM MobileFirst Platform Foundation in the following ways:

- The MobileFirst Operations Console, which is interactive.
- The **mfpadm** Ant task.
- The `mfpadm` program.
- The MobileFirst administration REST services.

The **mfpadm** Ant task, `mfpadm` 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, or
- Configuring a production server with the same settings as a test or preproduction server.

The **mfpadm** Ant task and the `mfpadm` program are simpler to use and have better error reporting than the REST services. The advantage of the **mfpadm** Ant task over the `mfpadm` program is that it is platform independent and easier to integrate when integration with Ant is already available.

Prerequisites

The mfpadm tool is installed with the MobileFirst Server installer. In the rest of this page, *product_install_dir* indicates the installation directory of the MobileFirst Server installer.

Apache Ant is required to run the **mfpadm** task. For information about the minimum supported version of Ant, see "System requirements" on page 2-7.

For convenience, Apache Ant 1.9.4 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 **mfpadm** Ant task on a different computer than the one on which you installed MobileFirst Server.
- Copy the file *product_install_dir*/MobileFirstServer/mfp-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 **mfpadm** Ant task, add this initialization command to the Ant script:

```
<taskdef resource="com/ibm/mfp/ant/deployers/antlib.xml">
  <classpath>
    <pathelement location="product_install_dir/MobileFirstServer/mfp-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

Other initialization commands that refer to the same mfp-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/ibm/mfp/ant/defaults.properties">
  <classpath>
    <pathelement location="product_install_dir/MobileFirstServer/mfp-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

For more information about running the MobileFirst Server installer, see "Running IBM Installation Manager" on page 6-40.

Calling the mfpadm Ant task

You can use the **mfpadm** Ant task and its associated commands to administer MobileFirst applications.

Syntax

Call the **mfpadm** Ant task as follows:

```
<mfpadm url=... user=... password=...|passwordfile=... [secure=...]>
    some commands
</mfpadm>
```

Attributes

The **`mfpadm`** Ant task has the following attributes:

Table 10-2. List of <mfpadm> 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	
`lockTimeout`	Timeout for acquiring a lock	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 **`mfpadm 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 **mfpadm** 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 **mfpadm** call fails.

Elements

You can use the following elements in **mfpadm** calls:

Table 10-3. Elements that can be used in <mfpadm>.

Element	Description	Count
`show-info`	Shows user and configuration information	0..∞
`show-global-config`	Shows global configuration information	0..∞
`show-diagnostics`	Shows diagnostics information	0..∞
`show-versions`	Shows versions information	0..∞
`unlock`	Releases the general-purpose lock	0..∞
`list-runtimes`	Lists the runtimes	0..∞
`show-runtime`	Shows information about a runtime	0..∞
`delete-runtime`	Deletes a runtime	0..∞
`show-user-config`	Shows the user configuration of a runtime	0..∞
`set-user-config`	Specifies the user configuration of a runtime	0..∞
`show-confidential-clients`	Shows the configurations of confidential clients of a runtime	0..∞
`set-confidential-clients`	Specifies the configurations of confidential clients of a runtime	0..∞
`set-confidential-clients-rule`	Specifies a rule for the confidential clients configuration of 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..∞

Table 10-3. Elements that can be used in <mfpadm> (continued).

Element	Description	Count
`show-app-version`	Shows information about an app version	0..∞
`delete-app-version`	Delete a version of an app	0..∞
`app`	Other operations on an app	0..∞
`app-version`	Other operations on an app version	0..∞
`list-devices`	Lists the devices	0..∞
`remove-device`	Removes a device	0..∞
`device`	Other operations for a device	0..∞
`list-farm-members`	Lists the members of the server farm	0..∞
`remove-farm-member`	Removes a server farm member	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`*`/MobileFirstServer/mfpadm-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 **`mfpadm`** 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 **`mfpadm`** command to a file.

Commands for general configuration

When you call the **`mfpadm`** Ant task, you can include various commands that access the global configuration of the IBM MobileFirst Platform Server or of a runtime.

The `show-global-config` command

The **`show-global-config`** command shows the global configuration. It has the following attributes:

Table 10-4. **`show-global-config`** *command attributes*

Attribute	Description	Required	Default
`output`	Name of the output file.	No	Not applicable
`outputproperty`	Name of the Ant property for the output.	No	Not applicable

Example

```
<show-global-config/>
```

This command is based on the "Global Configuration (GET)" on page 8-127 REST service.

The `show-user-config` command

The **`show-user-config`** command, outside of `<adapter>` and `<app-version>` elements, shows the user configuration of a runtime. It has the following attributes:

Table 10-5. **`show-user-config`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`format`	Specifies the output format. Either `json` or `xml`.	Yes	Not available
`output`	Name of the 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

```
<show-user-config runtime="mfp" format="xml"/>
```

This command is based on the "Runtime Configuration (GET)" on page 8-155 REST service.

The `set-user-config` command

The **`set-user-config`** command, outside of `<adapter>` and `<app-version>` elements, specifies the user configuration of a runtime. It has the following attributes for setting the entire configuration.

Table 10-6. **`set-user-config`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`file`	Name of the JSON or XML file that contains the new configuration.	Yes	Not available

The **`set-user-config`** command has the following attributes for setting a single property in the configuration.

Table 10-7. `set-user-config` command attributes

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`property`	Name of the JSON property. For a nested property, use the syntax `prop1.prop2.....propN`. For a JSON array element, use the index instead of a property name.	Yes	Not available
`value`	The value of the property.	Yes	Not available

Examples

```
<set-user-config runtime="mfp" file="myconfig.json"/>
<set-user-config runtime="mfp" property="timeout" value="240"/>
```

This command is based on the "Runtime configuration (PUT)" on page 8-157 REST service.

The `show-confidential-clients` command

The **`show-confidential-clients`** command shows the configuration of the confidential clients that can access a runtime. For more information about confidential clients, see "Confidential clients" on page 7-279. This command has the following attributes:

Table 10-8. `show-confidential-clients` command attributes

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`format`	Specifies the output format. Either `json` or `xml`.	Yes	Not available
`output`	Name of the 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

```
<show-confidential-clients runtime="mfp" format="xml" output="clients.xml"/>
```

This command is based on the "Confidential Clients (GET)" on page 8-62 REST service.

The `set-confidential-clients` command

The **`set-confidential-clients`** command specifies the configuration of the confidential clients that can access a runtime. For more information about confidential clients, see "Confidential clients" on page 7-279. This command has the following attributes:

*Table 10-9. **set-confidential-clients** command group attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime.	Yes	Not available
file	Name of the JSON or XML file that contains the new configuration.	Yes	Not available

Example

```
<set-confidential-clients runtime="mfp" file="clients.xml"/>
```

This command is based on the "Confidential Clients (PUT)" on page 8-64 REST service.

The `set-confidential-clients-rule` command

The **set-confidential-clients-rule** command specifies a rule in the configuration of the confidential clients that can access a runtime. For more information about confidential clients, see "Confidential clients" on page 7-279. This command has the following attributes:

*Table 10-10. **set-confidential-clients-rule** command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime.	Yes	Not available
id	The identifier of the rule.	Yes	Not available
displayName	The display name of the rule.	Yes	Not available
secret	The secret of the rule.	Yes	Not available
allowedScope	The scope of the rule. A space-separated list of tokens.	Yes	Not available

Example

```
<set-confidential-clients-rule runtime="mfp"
  id="push" displayName="Push" secret="lOa74Wxs" allowedScope="**"/>
```

This command is based on the "Confidential Clients (PUT)" on page 8-64 REST service.

Commands for adapters

When you call the **mfpadm** 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 10-11. **list-adapters** command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime.	Yes	Not available
output	Name of output file.	No	Not applicable

*Table 10-11. **list-adapters** command attributes (continued)*

Attribute	Description	Required	Default
outputproperty	Name of Ant property for the output.	No	Not applicable

Example

```
<list-adapters runtime="mfp"/>
```

This command is based on the “Adapters (GET)” on page 8-17 REST service.

The `deploy-adapter` command

The **deploy-adapter** command deploys an adapter in a runtime. It has the following attributes.

*Table 10-12. **deploy-adapter** command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime.	Yes	Not available
file	Binary adapter file (`.adapter`).	Yes	Not available

Example

```
<deploy-adapter runtime="mfp" file="MyAdapter.adapter"/>
```

This command is based on the “Adapter (POST)” on page 8-13 REST service.

The `show-adapter` command

The **show-adapter** command shows details about an adapter. It has the following attributes.

*Table 10-13. **show-adapter** command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime.	Yes	Not available
name	Name of an adapter.	Yes	Not available
output	Name of output file.	No	Not applicable
outputproperty	Name of Ant property for the output.	No	Not applicable

Example

```
<show-adapter runtime="mfp" name="MyAdapter"/>
```

This command is based on the “Adapter (GET)” on page 8-7 REST service.

The `delete-adapter` command

The **delete-adapter** command removes (undeploys) an adapter from a runtime. It has the following attributes.

Table 10-14. **`delete-adapter`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`name`	Name of an adapter.	Yes	Not available

Example

```
<delete-adapter runtime="mfp" name="MyAdapter"/>
```

This command is based on the "Adapter (DELETE)" on page 8-10 REST service.

The `adapter` command group

The **`adapter`** command group has the following attributes.

Table 10-15. **`adapter`** *command group attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`name`	Name of an adapter.	Yes	Not available

The **`adapter`** command supports the following elements.

Table 10-16. **`adapter`** *command group elements*

Element	Description	Count
`get-binary`	Gets the binary data.	0..∞
`show-user-config`	Shows the user configuration.	0..∞
`set-user-config`	Specifies the user configuration.	0..∞

The `get-binary` command

The **`get-binary`** command inside an `<adapter>` element returns the binary adapter file. It has the following attributes.

Table 10-17. **`get-binary`** *command attributes*

Attribute	Description	Required	Default
`tofile`	Name of the output file.	Yes	Not available

Example

```
<adapter runtime="mfp" name="MyAdapter">
  <get-binary tofile="/tmp/MyAdapter.adapter"/>
</adapter>
```

This command is based on the "Adapter (GET)" on page 8-7 REST service.

The `show-user-config` command

The **`show-user-config`** command, inside an `<adapter>` element, shows the user configuration of the adapter. It has the following attributes.

Table 10-18. **`show-user-config`** *command attributes*

Attribute	Description	Required	Default
`format`	Specifies the output format. Either `json` or `xml`.	Yes	Not available
`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

```
<adapter runtime="mfp" name="MyAdapter">
  <show-user-config format="xml"/>
</adapter>
```

This command is based on the "Adapter Configuration (GET)" on page 8-20 REST service.

The `set-user-config` command

The **`set-user-config`** command, inside an `<adapter>` element, specifies the user configuration of the adapter. It has the following attributes for setting the entire configuration.

Table 10-19. **`set-user-config`** *command attributes*

Attribute	Description	Required	Default
`file`	Name of the JSON or XML file that contains the new configuration.	Yes	Not available

The command has the following attributes for setting a single property in the configuration.

Table 10-20. **`set-user-config`** *command attributes*

Attribute	Description	Required	Default
`property`	Name of the JSON property. For a nested property, use the syntax `prop1.prop2.....propN`. For a JSON array element, use the index instead of a property name.	Yes	Not available
`value`	The value of the property.	Yes	Not available

Examples

```
<adapter runtime="mfp" name="MyAdapter">
  <set-user-config file="myconfig.json"/>
</adapter>
```

```
<adapter runtime="mfp" name="MyAdapter">
  <set-user-config property="timeout" value="240"/>
</adapter>
```

This command is based on the "Application Configuration (PUT)" on page 8-32 REST service.

Commands for apps

When you call the **mfpadm** Ant task, you can include various commands for apps.

The `list-apps` command

The **`list-apps`** command returns a list of the apps that are deployed in a runtime. It has the following attributes.

Table 10-21. **`list-apps`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`output`	Name of the output file.	No	Not applicable
`outputproperty`	Name of the Ant property for the output.	No	Not applicable

Example

```
<list-apps runtime="mfp"/>
```

This command is based on the "Applications (GET)" on page 8-48 REST service.

The `deploy-app` command

The **`deploy-app`** command deploys an app version in a runtime. It has the following attributes.

Table 10-22. **`deploy-app`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`file`	The application descriptor, a JSON file.	Yes	Not available

Example

```
<deploy-app runtime="mfp" file="MyApp/application-descriptor.json"/>
```

This command is based on the "Application (POST)" on page 8-43 REST service.

The `show-app` command

The **`show-app`** command returns a list of the app versions that are deployed in a runtime. It has the following attributes.

Table 10-23. **`show-app`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`name`	Name of an app.	Yes	Not available
`output`	Name of output file.	No	Not applicable
`outputproperty`	Name of Ant property for the output.	No	Not applicable

Example

```
<show-app runtime="mfp" name="MyApp"/>
```

This command is based on the "Application (GET)" on page 8-41 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 10-24. **`delete-app`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`name`	Name of an app.	Yes	Not available

Example

```
<delete-app runtime="mfp" name="MyApp"/>
```

This command is based on the "Application Version (DELETE)" on page 8-58 REST service.

The `show-app-version` command

The **`show-app-version`** command shows details about an app version in a runtime. It has the following attributes.

Table 10-25. **`show-app-version`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`name`	Name of the app.	Yes	Not available
`environment`	Mobile platform.	Yes	Not available
`version`	Version number of the app.	Yes	Not available

Example

```
<show-app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1"/>
```

This command is based on the "Application Version (GET)" on page 8-56 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 10-26. **`delete-app-version`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`name`	Name of an app.	Yes	Not available
`environment`	Mobile platform.	Yes	Not available

Table 10-26. **delete-app-version** *command attributes (continued)*

Attribute	Description	Required	Default
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="mfp" name="MyApp" environment="iphone" version="1.1"/>
```

This command is based on the "Application Version (DELETE)" on page 8-58 REST service.

The app command group

The **app** command group has the following attributes.

Table 10-27. **app** *command group attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime.	Yes	Not available
name	Name of an app.	Yes	Not available

The **app** command group supports the following elements.

Table 10-28. **app** *command group elements*

Element	Description	Count
show-license-config	Shows the token license configuration.	0..∞
set-license-config	Specifies the token license configuration.	0..∞
delete-license-config	Removes the token license configuration.	0..∞

The show-license-config command

The **show-license-config** command shows the token license configuration of an app. It has the following attributes.

Table 10-29. **show-license-config** *command attributes*

Attribute	Description	Required	Default
output	Name of a file in which to store the output.	Yes	Not available
outputproperty	Name of an Ant property in which to store the output.	Yes	Not available

Example

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <show-license-config output="/tmp/MyApp-license.xml"/>
</app-version>
```

This command is based on the "Application license configuration (GET)" on page 8-54 REST service.

The `set-license-config` command

The **`set-license-config`** command specifies the token license configuration of an app. It has the following attributes.

Table 10-30. **`set-license-config`** *command attributes*

Attribute	Description	Required	Default
`appType`	Type of app: `B2C` or `B2E`	Yes	Not available
`licenseType`	Type of application: `APPLICATION` or `ADDITIONAL_BRAND_DEPLOYMENT` or `NON_PRODUCTION`.	Yes	Not available

Example

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <set-license-config appType="B2E" licenseType="APPLICATION"/>
</app-version>
```

This command is based on the "Application License Configuration (POST)" on page 8-51 REST service.

The `delete-license-config` command

The **`delete-license-config`** command resets the token license configuration of an app, that is, reverts it to the initial state.

Example

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <delete-license-config/>
</app-version>
```

This command is based on the "License configuration (DELETE)" on page 8-136 REST service.

The `app-version` command group

The **`app-version`** command group has the following attributes.

Table 10-31. **`app-version`** *command group attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	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

The **`app-version`** command group supports the following elements:

Table 10-32. **app-version** *command group elements*

Element	Description	Count
get-descriptor	Gets the descriptor.	0..∞
get-web-resources	Gets the web resources.	0..∞
set-web-resources	Specifies the web resources.	0..∞
get-authenticity-data	Gets the authenticity data.	0..∞
set-authenticity-data	Specifies the authenticity data.	0..∞
delete-authenticity-data	Deletes the authenticity data.	0..∞
show-user-config	Shows the user configuration.	0..∞
set-user-config	Specifies the user configuration.	0..∞

The get-descriptor command

The **get-descriptor** command, inside an <app-version> element, returns the application descriptor of a version of an app. It has the following attributes.

Table 10-33. **get-descriptor** *command 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="mfp" name="MyApp" environment="iphone" version="1.1">
  <get-descriptor output="/tmp/MyApp-application-descriptor.json"/>
</app-version>
```

This command is based on the "Application Descriptor (GET)" on page 8-37 service.

The get-web-resources command

The **get-web-resources** command, inside an <app-version> element, returns the web resources of a version of an app, as a .zip file. It has the following attributes.

Table 10-34. **get-web-resources** *command attributes*

Attribute	Description	Required	Default
tofile	Name of the output file.	Yes	Not available

Example

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <get-web-resources tofile="/tmp/MyApp-web.zip"/>
</app-version>
```

This command is based on the "Retrieve Web Resource (GET)" on page 8-152 REST service.

The `set-web-resources` command

The **`set-web-resources`** command, inside an `<app-version>` element, specifies the web resources for a version of an app. It has the following attributes.

Table 10-35. **`set-web-resources`** *command attributes*

Attribute	Description	Required	Default
`file`	Name of the input file (must be a `.zip` file).	Yes	Not available

Example

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <set-web-resources file="/tmp/MyApp-web.zip"/>
</app-version>
```

This command is based on the "Deploy a web resource (POST)" on page 8-83 REST service.

The `get-authenticity-data` command

The **`get-authenticity-data`** command, inside an `<app-version>` element, returns the authenticity data of a version of an app. It has the following attributes.

Table 10-36. **`get-authenticity-data`** *command 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="mfp" name="MyApp" environment="iphone" version="1.1">
  <get-authenticity-data output="/tmp/MyApp.authenticity_data"/>
</app-version>
```

This command is based on the "Export runtime resources (GET)" on page 8-113 REST service.

The `set-authenticity-data` command

The **`set-authenticity-data`** command, inside an `<app-version>` element, specifies the authenticity data for a version of an app. It has the following attributes.

Table 10-37. **set-authenticity-data** *command attributes*

Attribute	Description	Required	Default
file	Name of the input file: • Either a authenticity_data file, • or a device file (.ipa, .apk, or .appx file), from which the authenticity data is extracted.	Yes	Not available

Examples

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <set-authenticity-data file="/tmp/MyApp.authenticity_data"/>
</app-version>
```

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <set-authenticity-data file="MyApp.ipa"/>
</app-version>
```

```
<app-version runtime="mfp" name="MyApp" environment="android" version="1.1">
  <set-authenticity-data file="MyApp.apk"/>
</app-version>
```

This command is based on the “Deploy Application Authenticity Data (POST)” on page 8-80 REST service.

The `delete-authenticity-data` command

The **delete-authenticity-data** command, inside an <app-version> element, deletes the authenticity data of a version of an app. It has no attributes.

Example

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <delete-authenticity-data/>
</app-version>
```

This command is based on the “Application Authenticity (DELETE)” on page 8-27 REST service.

The `show-user-config` command

The **show-user-config** command, inside an <app-version> element, shows the user configuration of a version of an app. It has the following attributes.

Table 10-38. **show-user-config** *command attributes*

Attribute	Description	Required	Default
format	Specifies the output format. Either json or xml.	Yes	Not available
output	Name of the output file.	No	Not applicable
outputproperty	Name of the Ant property for the output.	No	Not applicable

Examples

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <show-user-config format="json" output="/tmp/MyApp-config.json"/>
</app-version>
```

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <show-user-config format="xml" output="/tmp/MyApp-config.xml"/>
</app-version>
```

This command is based on the “Application Configuration (GET)” on page 8-30 REST service.

The `set-user-config` command

The **`set-user-config`** command, inside an `<app-version>` element, specifies the user configuration for a version of an app. It has the following attributes for setting the entire configuration.

Table 10-39. **`set-user-config`** *command attributes*

Attribute	Description	Required	Default
`file`	Name of the JSON or XML file that contains the new configuration.	Yes	Not available

The **`set-user-config`** command has the following attributes for setting a single property in the configuration.

Table 10-40. **`set-user-config`** *command attributes*

Attribute	Description	Required	Default
`property`	Name of the JSON property. For a nested property, use the syntax `prop1.prop2.....propN`. For a JSON array element, use the index instead of a property name.	Yes	Not available
`value`	The value of the property.	Yes	Not available

Examples

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <set-user-config file="/tmp/MyApp-config.json"/>
</app-version>
```

```
<app-version runtime="mfp" name="MyApp" environment="iphone" version="1.1">
  <set-user-config property="timeout" value="240"/>
</app-version>
```

This command is based on the “Application Configuration (PUT)” on page 8-32 REST service.

Commands for devices

When you call the **`mfpadm`** 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 10-41. `list-devices` command attributes

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`query`	A friendly name or user identifier to search for. This 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.	No	Not applicable
`output`	Name of output file.	No	Not applicable
`outputproperty`	Name of Ant property for the output.	No	Not applicable

Examples

```
<list-devices runtime="mfp"/>
```

```
<list-devices runtime="mfp" query="john"/>
```

This command is based on the "Devices (GET)" on page 8-96 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 10-42. `remove-device` command attributes

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`id`	Unique device identifier.	Yes	Not available

Example

```
<remove-device runtime="mfp" id="496E974CCEDE86791CF9A8EF2E5145B6"/>
```

This command is based on the "Device (DELETE)" on page 8-93 REST service.

The `device` command group

The **`device`** command group has the following attributes.

Table 10-43. `device` command group attributes

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`id`	Unique device identifier.	Yes	Not available

The **`device`** command supports the following elements.

Table 10-44. `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 10-45. `set-status` command attributes

Attribute	Description	Required	Default
`status`	New status.	Yes	Not available

The status can have one of the following values:
- `ACTIVE`
- `LOST`
- `STOLEN`
- `EXPIRED`
- `DISABLED`

Example

```
<device runtime="mfp" id="496E974CCEDE86791CF9A8EF2E5145B6">
  <set-status status="EXPIRED"/>
</device>
```

This command is based on the “Device Status (PUT)” on page 8-90 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 10-46. `set-appstatus` command attributes

Attribute	Description	Required	Default
`app`	Name of an app.	Yes	Not available
`status`	New status.	Yes	Not available

The status can have one of the following values:

- `ENABLED`
- `DISABLED`

Example

```
<device runtime="mfp" id="496E974CCEDE86791CF9A8EF2E5145B6">
  <set-appstatus app="MyApp" status="DISABLED"/>
</device>
```

This command is based on the "Device Application Status (PUT)" on page 8-86 REST service.

Commands for troubleshooting

You can use Ant task commands to investigate problems with 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. Use this command to test whether the MobileFirst administration services are running at all. It has the following attributes:

Table 10-47. **`show-info`** *command attributes*

Attribute	Description	Required	Default
`output`	Name of output file.	No	Not applicable
`outputproperty`	Name of Ant property for the output.	No	Not applicable

Example

```
<show-info/>
```

The `show-versions` command

The **`show-versions`** command displays the MobileFirst versions of various components:

- **`mfpadmVersion`**: the exact MobileFirst Server version number from which the `mfp-ant-deployer.jar` file is taken.
- **`productVersion`**: the exact MobileFirst Server version number from which the `mfp-admin-service.war` file is taken.
- **`mfpAdminVersion`**: the exact build version number of `mfp-admin-service.war` alone.

The command has the following attributes:

Table 10-48. **`show-versions`** *command attributes*

Attribute	Description	Required	Default
`output`	Name of output file.	No	Not applicable
`outputproperty`	Name of Ant property for the output.	No	Not applicable

Example

```
<show-versions/>
```

The show-diagnostics command

The **show-diagnostics** command shows the status of various components that are necessary for the correct operation of the MobileFirst administration service, such as the availability of the database and of auxiliary services. This command has the following attributes.

Table 10-49. **show-diagnostics** *command attributes*

Attribute	Description	Required	Default
output	Name of output file.	No	Not applicable
outputproperty	Name of Ant property for the output.	No	Not applicable

Example

```
<show-diagnostics/>
```

The `unlock` command

The **unlock** command releases the general-purpose lock. Some destructive operations take this lock in order to prevent concurrent modification of the same configuration data. In rare cases, if such an operation is interrupted, the lock might remain in locked state, making further destructive operations impossible. Use the **unlock** command to release the lock in such situations. The command has no attributes.

Example

```
<unlock/>
```

The `list-runtimes` command

The **list-runtimes** command returns a list of the deployed runtimes. It has the following attributes:

Table 10-50. **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	Not applicable
outputproperty	Name of Ant property for the output.	No	Not applicable

Examples

```
<list-runtimes/>
<list-runtimes inDatabase="true"/>
```

This command is based on the "Runtimes (GET)" on page 8-170 REST service.

The `show-runtime` command

The **`show-runtime`** command shows information about a given deployed runtime. It has the following attributes:

Table 10-51. **`show-runtime`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`output`	Name of output file.	No	Not applicable
`outputproperty`	Name of Ant property for the output.	No	Not applicable

Example

```
<show-runtime runtime="mfp"/>
```

This command is based on the "Runtime (GET)" on page 8-161 REST service.

The `delete-runtime` command

The **`delete-runtime`** command deletes the runtime, including its apps and adapters, from the database. You can delete a runtime only when its web application is stopped. The command has the following attributes.

Table 10-52. **`delete-runtime`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`condition`	Condition when to delete it: `empty` or `always` **Attention:** The `always` option is dangerous.	No	Not applicable

Example

```
<delete-runtime runtime="mfp" condition="empty"/>
```

This command is based on the "Runtime (DELETE)" on page 8-166 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 10-53. **`list-farm-members`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	Yes	Not available
`output`	Name of output file.	No	Not applicable
`outputproperty`	Name of Ant property for the output.	No	Not applicable

Example

```
<list-farm-members runtime="mfp"/>
```

This command is based on the “Farm topology members (GET)” on page 8-114 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. Use this command when the server has become unavailable or disconnected. The command has the following attributes.

Table 10-54. **`remove-farm-member`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime.	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="mfp" serverId="srvlx15"/>
```

This command is based on the “Farm topology members (DELETE)” on page 8-116 REST service.

Administering MobileFirst applications through the command line

You can administer MobileFirst applications through the `mfpadm` program.

Comparison with other facilities

You can run administration operations with IBM MobileFirst Platform Foundation in the following ways:

- The MobileFirst Operations Console, which is interactive.
- The **`mfpadm`** Ant task.
- The `mfpadm` program.
- The MobileFirst administration REST services.

The **`mfpadm`** Ant task, `mfpadm` program, and REST services are useful for automated or unattended execution of operations, such as the following use cases:

- Eliminating operator errors in repetitive operations, or
- Operating outside the operator's normal working hours, or
- Configuring a production server with the same settings as a test or preproduction server.

The `mfpadm` program and the **`mfpadm`** Ant task are simpler to use and have better error reporting than the REST services. The advantage of the `mfpadm` program over the **`mfpadm`** 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 `mfpadm` tool is installed with the MobileFirst Server installer. In the rest of this page, *product_install_dir* indicates the installation directory of the MobileFirst Server installer.

The **`mfpadm`** command is provided in the *`product_install_dir`*`/shortcuts/` directory as a set of scripts:

- `mfpadm` for UNIX / Linux
- `mfpadm.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 `mfpadm` program, either put the *`product_install_dir`*`/shortcuts/` directory into your `PATH` environment variable, or reference its absolute file name in each call.

For more information about running the MobileFirst Server installer, see "Running IBM Installation Manager" on page 6-40.

Calling the mfpadm program

You can use the `mfpadm` program to administer MobileFirst applications.

Syntax

Call the `mfpadm` program as follows:

```
mfpadm --url= --user= ... [--passwordfile=...] [--secure=false] some command
```

The `mfpadm` program has the following options:

Table 10-55. mfpadm 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	

Table 10-55. mfpadm program options (continued)

Option	Type	Description	Required	Default
--connection-request-timeout	Number	Timeout for obtaining an entry from a connection request pool, in seconds	No	
--lock-timeout	Number	Timeout for acquiring a lock, in seconds	No	2
--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/mfpadmin`
- For Tomcat: `https://server:8443/mfpadmin`

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 10-51), 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 `mfpadm` calls contains a command. The following commands are supported.

Table 10-56. mfpadm invocation supported commands

Command	Description
`show info`	Shows user and configuration information.
`show global-config`	Shows global configuration information.
`show diagnostics`	Shows diagnostics information.
`show versions`	Shows version information.
`unlock`	Releases the general-purpose lock.
`list runtimes [--in-database]`	Lists the runtimes.

Table 10-56. mfpadm invocation supported commands (continued)

Command	**Description**
show runtime [*runtime-name*]	Shows information about a runtime.
delete runtime [*runtime-name*] *condition*	Deletes a runtime.
show user-config [*runtime-name*]	Shows the user configuration of a runtime.
set user-config [*runtime-name*] *file*	Specifies the user configuration of a runtime.
set user-config [*runtime-name*] *property* = *value*	Specifies a property in the user configuration of a runtime.
show confidential-clients [*runtime-name*]	Shows the configuration of the confidential clients of a runtime.
set confidential-clients [*runtime-name*] *file*	Specifies the configuration of the confidential clients of a runtime.
set confidential-clients-rule [*runtime-name*] *id display-name secret allowed-scope*	Specifies a rule for the configuration of the confidential clients of a runtime.
list adapters [*runtime-name*]	Lists the adapters.
deploy adapter [*runtime-name*] *property* = *value*	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.
show app version [*runtime-name*] *app-name environment version*	Shows information about an app version.
delete app version [*runtime-name*] *app-name environment version*	Deletes a version of an app.
app [*runtime-name*] *app-name* show license-config	Shows the token license configuration of an app.
app [*runtime-name*] *app-name* set license-config *app-type license-type*	Specifies the token license configuration for an app.
app [*runtime-name*] *app-name* delete license-config	Removes the token license configuration for an app.
app version [*runtime-name*] *app-name environment version* get descriptor [> *tofile*]	Gets the descriptor of an app version.
app version [*runtime-name*] *app-name environment version* get web-resources [> *tofile*]	Gets the web resources of an app version.
app version [*runtime-name*] *app-name environment version* set web-resources *file*	Specifies the web resources of an app version.

Table 10-56. mfpadm invocation supported commands (continued)

Command	Description
`app version [`*runtime-name*`]` *app-name* *environment* *version* `get authenticity-data [>` *tofile*`]`	Gets the authenticity data of an app version.
`app version [`*runtime-name*`]` *app-name* *environment* *version* `set authenticity-data [`*file*`]`	Specifies the authenticity data of an app version.
`app version [`*runtime-name*`]` *app-name* *environment* *version* `delete authenticity-data`	Deletes the authenticity data of an app version.
`app version [`*runtime-name*`]` *app-name* *environment* *version* `show user-config`	Shows the user configuration of an app version.
`app version [`*runtime-name*`]` *app-name* *environment* *version* `set user-config` *file*	Specifies the user configuration of an app version.
`app version [`*runtime-name*`]` *app-name* *environment* *version* `set user-config` *property* = *value*	Specifies a property in the user configuration of an app version.
`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.
`list farm-members [`*runtime-name*`]`	Lists the servers that are members of the server farm.
`remove farm-member [`*runtime-name*`]` *server-id*	Removes a server from the list of farm members.

Interactive mode

Alternatively, you can also call `mfpadm` 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 `mfpadm`.

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 `mfpadm` 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`
3. 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`*
- `--lock-timeout=`*`seconds`*
- *runtime-name*

Use these commands to store these values in the configuration file.

Table 10-57. Commands to store values in the configuration file

Command	Comment
`mfpadm [--configfile=`*`file`*`] config url` *`URL`*	
`mfpadm [--configfile=`*`file`*`] config secure` *`boolean`*	
`mfpadm [--configfile=`*`file`*`] config user` *`name`*	

Table 10-57. Commands to store values in the configuration file (continued)

Command	Comment
`mfpadm [--configfile=`*file*`] config password`	Prompts for the password.
`mfpadm [--configfile=`*file*`] config timeout` *seconds*	
`mfpadm [--configfile=`*file*`] config connect-timeout` *seconds*	
`mfpadm [--configfile=`*file*`] config socket-timeout` *seconds*	
`mfpadm [--configfile=`*file*`] config connection-request-timeout` *seconds*	
`mfpadm [--configfile=`*file*`] config lock-timeout` *seconds*	
`mfpadm [--configfile=`*file*`] config runtime` *runtime-name*	

Use this command to list the values that are stored in the configuration file: `mfpadm [--configfile=`*file*`] config`

The configuration file is a text file, in the encoding of the current locale, in Java `.properties` syntax. Default configuration file:
- UNIX: `$HOME/.mfpadm.config`
- Windows: `My Documents\IBM MobileFirst Platform Server Data\mfpadm.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 10-58. 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 `mfpadm` 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 **`mfpadm`** Ant task, with attribute `output="`*`some file name`*`"` to redirect the output of a command to a file.

Commands for general configuration

When you call the **`mfpadm`** program, you can include various commands that access the global configuration of the IBM MobileFirst Platform Server or of a runtime.

The `show global-config` command

The **`show global-config`** command shows the global configuration.

Syntax: **show global-config**

It takes the following options:

Table 10-59. **`show global-config`** *options*

Argument	Description
`--xml`	Produces XML output instead of tabular output.

Example

```
show global-config
```

This command is based on the “Global Configuration (GET)” on page 8-127 REST service.

The `show user-config` command

The **`show user-config`** command shows the user configuration of a runtime.

Syntax: **show user-config [--xml] [runtime-name]**

It takes the following arguments:

Table 10-60. **`show user-config`** *arguments*

Argument	Description
`runtime-name`	Name of the runtime.

The **`show user-config`** command takes the following options after the verb.

Table 10-61. **`show user-config`** *options*

Argument	Description	Required	Default
`--xml`	Produces output in XML format instead of JSON format.	No	Standard output

Example

```
show user-config mfp
```

This command is based on the "Runtime Configuration (GET)" on page 8-155 REST service.

The `set user-config` command

The `set user-config` command specifies the user configuration of a runtime or a single property among this configuration.

Syntax for the entire configuration: **set user-config [runtime-name] file**

It takes the following arguments:

Table 10-62. **`set user-config`** *arguments*

Attribute	Description
`runtime-name`	Name of the runtime.
`file`	Name of the JSON or XML file that contains the new configuration.

Syntax for a single property: **set user-config [runtime-name] property = value**

The `set user-config` command takes the following arguments:

Table 10-63. **`set user-config`** *arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`property`	Name of the JSON property. For a nested property, use the syntax `prop1.prop2.....propN`. For a JSON array element, use the index instead of a property name.
`value`	The value of the property.

Examples

```
set user-config mfp myconfig.json
set user-config mfp timeout = 240
```

This command is based on the "Runtime configuration (PUT)" on page 8-157 REST service.

The `show confidential-clients` command

The `show confidential-clients` command shows the configuration of the confidential clients that can access a runtime. For more information about confidential clients, see “Confidential clients” on page 7-279.

Syntax: **show confidential-clients [--xml] [runtime-name]**

It takes the following arguments:s

Table 10-64. **`show confidential-clients`** *arguments*

Attribute	Description
`runtime-name`	Name of the runtime.

The `show confidential-clients` command takes the following options after the verb.

Table 10-65. **`show confidential-clients`** *options*

Argument	Description	Required	Default
`--xml`	Produces output in XML format instead of JSON format.	No	Standard output

Example

```
show confidential-clients --xml mfp
```

This command is based on the “Confidential Clients (GET)” on page 8-62 REST service.

The `set confidential-clients` command

The `set confidential-clients` command specifies the configuration of the confidential clients that can access a runtime. For more information about confidential clients, see “Confidential clients” on page 7-279.

Syntax: **set confidential-clients [runtime-name] file**

Its takes the following arguments:

Table 10-66. **`set confidential-clients`** *arguments*

Attribute	Description
`runtime-name`	Name of the runtime.
`file`	Name of the JSON or XML file that contains the new configuration.

Example

```
set confidential-clients mfp clients.xml
```

This command is based on the “Confidential Clients (PUT)” on page 8-64 REST service.

The `set confidential-clients-rule` command

The `set confidential-clients-rule` command specifies a rule in the configuration of the confidential clients that can access a runtime. For more information about confidential clients, see "Confidential clients" on page 7-279.

Syntax: **set confidential-clients-rule [runtime-name] id displayName secret allowedScope**

It takes the following arguments:

Table 10-67. set confidential-clients-rule arguments

Attribute	Description
`runtime`	Name of the runtime.
`id`	The identifier of the rule.
`displayName`	The display name of the rule.
`secret`	The secret of the rule.
`allowedScope`	The scope of the rule. A space-separated list of tokens. Use double-quotes to pass a list of two or more tokens.

Example

```
set confidential-clients-rule mfp push Push lOa74Wxs "**"
```

This command is based on the "Confidential Clients (PUT)" on page 8-64 REST service.

Commands for adapters

When you invoke the `mfpadm` 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 10-68. **`list adapters`** command arguments*

Argument	Description
`runtime-name`	Name of the runtime.

The **`list adapters`** command takes the following options after the object.

*Table 10-69. **`list adapters`** options*

Option	Description
`--xml`	Produce XML output instead of tabular output.

Example

```
list adapters mfp
```

This command is based on the “Adapters (GET)” on page 8-17 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 10-70. **`deploy adapter`** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`file`	Binary adapter file (`.adapter`)

Example

```
deploy adapter mfp MyAdapter.adapter
```

This command is based on the “Adapter (POST)” on page 8-13 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 10-71. **`show adapter`** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`adapter-name`	Name of an adapter

The **`show adapter`** command takes the following options after the object.

Table 10-72. **`show adapter`** *options*

Option	Description
`--xml`	Produce XML output instead of tabular output.

Example

```
show adapter mfp MyAdapter
```

This command is based on the “Adapter (GET)” on page 8-7 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 10-73. **`delete adapter`** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`adapter-name`	Name of an adapter.

Example

```
delete adapter mfp MyAdapter
```

This command is based on the "Adapter (DELETE)" on page 8-10 REST service.

The `adapter` command prefix

The **`adapter`** command prefix takes the following arguments before the verb.

Table 10-74. **`adapter`** *command prefix arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`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 10-75. **`adapter get binary`** *options*

Option	Description	Required	Default
`> tofile`	Name of the output file.	No	Standard output

Example

```
adapter mfp MyAdapter get binary > /tmp/MyAdapter.adapter
```

This command is based on the "Export runtime resources (GET)" on page 8-113 REST service.

The `adapter show user-config` command

The **`adapter show user-config`** command shows the user configuration of the adapter.

Syntax: **adapter [runtime-name] adapter-name show user-config [--xml]**

The **adapter show user-config** command takes the following options after the verb.

Table 10-76. **adapter show user-config** *options*

Option	Description
`--xml`	Produces output in XML format instead of JSON format.

Example

```
adapter mfp MyAdapter show user-config
```

This command is based on the “Adapter Configuration (GET)” on page 8-20 REST service.

The adapter set user-config command

The **adapter set user-config** command specifies the user configuration of the adapter or a single property within this configuration.

Syntax for the entire configuration: **adapter [runtime-name] adapter-name set user-config file**

The **adapter set user-config** command takes the following arguments after the verb.

Table 10-77. **adapter set user-config** *arguments*

Option	Description
`file`	Name of the JSON or XML file that contains the new configuration.

Syntax for a single property: **adapter [runtime-name] adapter-name set user-config property = value**

It takes the following arguments after the verb.

Table 10-78. **adapter set user-config** *arguments*

Option	Description
`property`	Name of the JSON property. For a nested property, use the syntax `prop1.prop2.....propN`. For a JSON array element, use the index instead of a property name.
`value`	The value of the property.

Examples

```
adapter mfp MyAdapter set user-config myconfig.json
```

```
adapter mfp MyAdapter set user-config timeout = 240
```

This command is based on the “Adapter configuration (PUT)” on page 8-22 REST service.

Commands for apps

When you invoke the `mfpadm` program, you can include various commands for apps.

The `list apps` command

The **`list apps`** command returns a list of the apps that are deployed in a runtime.

Syntax: **list apps [runtime-name]**

It takes the following arguments:

*Table 10-79. **`list apps`** command arguments*

Argument	Description
`runtime-name`	Name of the runtime.

The **`list apps`** command takes the following options after the object.

*Table 10-80. **`list apps`** command options*

Option	Description
`--xml`	Produce XML output instead of tabular output.

Example

```
list apps mfp
```

This command is based on the "Applications (GET)" on page 8-48 REST service.

The `deploy app` command

The **`deploy app`** command deploys an app version in a runtime.

Syntax: **deploy app [runtime-name] file**

It takes the following arguments:

*Table 10-81. **`deploy app`** command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`file`	The application descriptor, a JSON file.

Example

```
deploy app mfp MyApp/application-descriptor.json
```

This command is based on the "Application (POST)" on page 8-43 REST service.

The `show app` command

The **`show app`** command shows details about an app in a runtime, in particular its environments and versions.

Syntax: **show app [runtime-name] app-name**

It takes the following arguments:

Table 10-82. **`show app`** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`app-name`	Name of an app.

The **`show app`** command takes the following options after the object.

Table 10-83. **`show app`** *command options*

Option	Description
`--xml`	Produce XML output instead of tabular output.

Example

```
show app mfp MyApp
```

This command is based on the “Application (GET)” on page 8-41 REST service.

The `delete app` command

The **`delete app`** command removes (undeploys) an app, from all environments and all versions, from a runtime. 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 10-84. **`delete app`** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`app-name`	Name of an app

Example

```
delete app mfp MyApp
```

This command is based on the “Application Version (DELETE)” on page 8-58 REST service.

The `show app version` command

The **`show app version`** command show details about an app version in a runtime.

Syntax: **show app version [runtime-name] app-name environment version**

It takes the following arguments:

Table 10-85. **show app** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`app-name`	Name of an app.
`environment`	Mobile platform.
`version`	Version of the app.

The **show app version** command takes the following options after the object.

Table 10-86. **show app** *command options*

Argument	Description
`--xml`	Produces XML output instead of tabular output.

Example

```
show app version mfp MyApp iPhone 1.1
```

This command is based on the "Application Version (GET)" on page 8-56 REST service.

The `delete app version` command

The **delete app version** command removes (undeploys) an app version from a runtime.

Syntax: **delete app version [runtime-name] app-name environment version**

It takes the following arguments:

Table 10-87. **delete app version** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`app-name`	Name of an app.
`environment`	Mobile platform.
`version`	Version of the app.

Example

```
delete app version mfp MyApp iPhone 1.1
```

This command is based on the "Application Version (DELETE)" on page 8-58 REST service.

The `app` command prefix

The **app** command prefix takes the following arguments before the verb.

Table 10-88. **app** *command prefix arguments*

Argument	Description
`runtime-name`	Name of the runtime.

Table 10-88. **app** *command prefix arguments (continued)*

Argument	Description
app-name	Name of an app.

The `app show license-config` command

The **app show license-config** command shows the token license configuration of an app.

Syntax: **app [runtime-name] app-name show license-config**

It takes the following options after the object:

Table 10-89. **app show license-config** *option*

Argument	Description
--xml	Produces XML output instead of tabular output.

Example

```
app mfp MyApp show license-config
```

This command is based on the "Application license configuration (GET)" on page 8-54 REST service.

The `app set license-config` command

The **app set license-config** command specifies the token license configuration of an app.

Syntax: **app [runtime-name] app-name set license-config app-type license-type**

It takes the following arguments after the verb.

Table 10-90. **app set license-config** *command arguments*

Argument	Description
appType	Type of app: B2C or B2E.
licenseType	Type of application: APPLICATION or ADDITIONAL_BRAND_DEPLOYMENT or NON_PRODUCTION.

Example

```
app mfp MyApp iPhone 1.1 set license-config B2E APPLICATION
```

This command is based on the "Application License Configuration (POST)" on page 8-51 REST service.

The `app delete license-config` command

The **app delete license-config** command resets the token license configuration of an app, that is, reverts it to the initial state.

Syntax: **app [runtime-name] app-name delete license-config**

Example

```
app mfp MyApp iPhone 1.1 delete license-config
```

This command is based on the “License configuration (DELETE)” on page 8-136 REST service.

The `app version` command prefix

The **`app version`** command prefix takes the following arguments before the verb.

Table 10-91. **`app version`** *command prefix arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`app-name`	Name of an app.
`environment`	Mobile platform
`version`	Version of the app

The `app version get descriptor` command

The **`app version get descriptor`** command returns the application descriptor of a version of an app.

Syntax: **app version [runtime-name] app-name environment version get descriptor [> tofile]**

It takes the following arguments after the verb.

Table 10-92. **`app version get descriptor`** *command options*

Argument	Description	Required	Default
`> tofile`	Name of the output file.	No	Standard output

Example

```
app version mfp MyApp iPhone 1.1 get descriptor > /tmp/MyApp-application-descriptor.json
```

This command is based on the “Application Descriptor (GET)” on page 8-37 REST service.

The `app version get web-resources` command

The **`app version get web-resources`** command returns the web resources of a version of an app, as a `.zip` file.

Syntax: **app version [runtime-name] app-name environment version get web-resources [> tofile]**

It takes the following arguments after the verb.

Table 10-93. **`app version get web-resources`** *command options*

Argument	Description	Required	Default
`> tofile`	Name of the output file.	No	Standard output

Example

```
app version mfp MyApp iPhone 1.1 get web-resources > /tmp/MyApp-web.zip
```

This command is based on the "Retrieve Web Resource (GET)" on page 8-152 REST service.

The `app version set web-resources` command

The **`app version set web-resources`** command specifies the web resources for a version of an app.

Syntax: **app version [runtime-name] app-name environment version set web-resources file**

It takes the following arguments after the verb.

Table 10-94. **`app version set web-resources`** *command arguments*

Argument	Description
`file`	Name of the input file (must be a `.zip` file).

Example

```
app version mfp MyApp iPhone 1.1 set web-resources /tmp/MyApp-web.zip
```

This command is based on the "Deploy a web resource (POST)" on page 8-83 REST service.

The `app version get authenticity-data` command

The **`app version get authenticity-data`** command returns the authenticity data of a version of an app.

Syntax: **app version [runtime-name] app-name environment version get authenticity-data [> tofile]**

It takes the following arguments after the verb.

Table 10-95. **`app version get authenticity-data`** *command options*

Argument	Description	Required	Default	
`> tofile`	Name of the output file.	No	Standard output	

Example

```
app version mfp MyApp iPhone 1.1 get authenticity-data > /tmp/MyApp.authenticity_data
```

This command is based on the Export runtime resources (GET) REST service.

The `app version set authenticity-data` command

The **`app version set authenticity-data`** command specifies the authenticity data for a version of an app.

Syntax: **app version [runtime-name] app-name environment version set authenticity-data file**

It takes the following arguments after the verb.

Table 10-96. **`app version set authenticity-data`** *command arguments*

Argument	Description
`file`	Name of the input file: • Either a `.authenticity_data` file, • Or a device file (`.ipa` or `.apk` or `.appx`), from which the authenticity data is extracted.

Examples

```
app version mfp MyApp iPhone 1.1 set authenticity-data /tmp/MyApp.authenticity_data
app version mfp MyApp iPhone 1.1 set authenticity-data MyApp.ipa
app version mfp MyApp android 1.1 set authenticity-data MyApp.apk
```

This command is based on the "Deploy Application Authenticity Data (POST)" on page 8-80 REST service.

The `app version delete authenticity-data` command

The **`app version delete authenticity-data`** command deletes the authenticity data for a version of an app.

Syntax: **app version [runtime-name] app-name environment version delete authenticity-data**

Example

```
app version mfp MyApp iPhone 1.1 delete authenticity-data
```

This command is based on the "Application Authenticity (DELETE)" on page 8-27 REST service.

The `app version show user-config` command

The **`app version show user-config`** command shows the user configuration of a version of an app.

Syntax: **app version [runtime-name] app-name environment version show user-config [--xml]**

It takes the following options after the verb.

Table 10-97. **`app version show user-config`** *command options*

Argument	Description	Required	Default
`[--xml]`	Produce output in XML format instead of JSON format.	No	Standard output

Example

```
app version mfp MyApp iPhone 1.1 show user-config
```

This command is based on the "Application Configuration (GET)" on page 8-30 REST service.

The `app version set user-config` command

The **`app version set user-config`** command specifies the user configuration for a version of an app or a single property among this configuration.

Syntax for the entire configuration: **app version [runtime-name] app-name environment version set user-config file**

It takes the following arguments after the verb.

Table 10-98. **`app version set user-config`** *command arguments*

Argument	Description
`file`	Name of the JSON or XML file that contains the new configuration.

Syntax for a single property: **app version [runtime-name] app-name environment version set user-config property = value**

The **`app version set user-config`** command takes the following arguments after the verb.

Table 10-99. **`app version set user-config`** *command arguments*

Argument	Description
`property`	Name of the JSON property. For a nested property, use the syntax `prop1.prop2.....propN`. For a JSON array element, use the index instead of a property name.
`value`	The value of the property.

Examples

```
app version mfp MyApp iPhone 1.1 set user-config /tmp/MyApp-config.json
app version mfp MyApp iPhone 1.1 set user-config timeout = 240
```

This command is based on the "Application Configuration (PUT)" on page 8-32 REST service.

Commands for devices

When you invoke the `mfpadm` 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 10-100. **list devices** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`query`	A friendly name or user identifier, to search for. This 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.

The **list devices** command takes the following options after the object.

Table 10-101. **list devices** *command options*

Option	Description
`--xml`	Produces XML output instead of tabular output.

Examples

```
list-devices mfp
```

```
list-devices mfp --query=john
```

This command is based on the "Devices (GET)" on page 8-96 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 10-102. **remove device** *command arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`id`	Unique device identifier.

Example

```
remove device mfp 496E974CCEDE86791CF9A8EF2E5145B6
```

This command is based on the "Device (DELETE)" on page 8-93 REST service.

The `device` command prefix

The **device** command prefix takes the following arguments before the verb.

Table 10-103. **device** *command prefix arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`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 10-104. **`device set status`** *command arguments*

Argument	Description
`new-status`	New status.

The status can have one of the following values:
- `ACTIVE`
- `LOST`
- `STOLEN`
- `EXPIRED`
- `DISABLED`

Example

```
device mfp 496E974CCEDE86791CF9A8EF2E5145B6 set status EXPIRED
```

This command is based on the "Device Status (PUT)" on page 8-90 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 10-105. **`device set appstatus`** *command arguments*

Argument	Description
`app-name`	Name of an app.
`new-status`	New status.

The status can have one of the following values:
- `ENABLED`
- `DISABLED`

Example

```
device mfp 496E974CCEDE86791CF9A8EF2E5145B6 set appstatus MyApp DISABLED
```

This command is based on the "Device Application Status (PUT)" on page 8-86 REST service.

Commands for troubleshooting

When you invoke the `mfpadm` 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**

The **`show info`** command takes the following options after the object.

Table 10-106. **`show infos`** *options*

Option	Description
`--xml`	Produces XML output instead of tabular output.

Example

```
show info
```

The `show versions` command

The **`show versions`** command displays the MobileFirst versions of various components:

- `mfpadmVersion`: the exact MobileFirst Server version number from which `mfp-ant-deployer.jar` is taken.
- `productVersion`: the exact MobileFirst Server version number from which `mfp-admin-service.war` is taken
- `mfpAdminVersion`: the exact build version number of `mfp-admin-service.war` alone.

Syntax: **show versions**

The **`show versions`** command takes the following options after the object.

Table 10-107. **`show versions`** *options*

Option	Description
`--xml`	Produces XML output instead of tabular output.

Example

```
show versions
```

The `show diagnostics` command

The **`show diagnostics`** command shows the status of various components that are necessary for the correct operation of the MobileFirst administration service, such as the availability of the database and of auxiliary services.

Syntax: **show diagnostics**

The **`show diagnostics`** command takes the following options after the object.

Table 10-108. **show diagnostics** *options*

Option	Description
--xml	Produces XML output instead of tabular output.

Example

```
show diagnostics
```

The unlock command

The **unlock** command releases the general-purpose lock. Some destructive operations take this lock in order to prevent concurrent modification of the same configuration data. In rare cases, if such an operation is interrupted, the lock might remain in locked state, making further destructive operations impossible. Use the **unlock** command to release the lock in such situations.

Example

```
unlock
```

The list runtimes command

The **list runtimes** command returns a list of the deployed runtimes.

Syntax: **list runtimes [--in-database]**

The **list runtimes** command takes the following options:

Table 10-109. **list runtimes** *options*

Option	Description
--in-database	Whether to look in the database instead of via MBeans
--xml	Produces XML output instead of tabular output.

Examples

```
list runtimes
list runtimes --in-database
```

This command is based on the “Runtimes (GET)” on page 8-170 REST service.

The show runtime command

The **show runtime** command shows information about a given deployed runtime.

Syntax: **show runtime [runtime-name]**

The **show runtime** command takes the following arguments:

Table 10-110. The **show runtime** *command arguments*

Argument	Description
runtime-name	Name of the runtime.

The **show runtime** command takes the following options after the object.

Table 10-111. The **show runtime** *command options*

Option	Description
`--xml`	Produces XML output instead of tabular output.

This command is based on the "Runtime (GET)" on page 8-161 REST service.

Example

```
show runtime mfp
```

The `delete runtime` command

The **delete runtime** command deletes a runtime, including its apps and adapters, from the database. You can delete a runtime only when its web application is stopped.

Syntax: **delete runtime [runtime-name] condition**

The **delete runtime** command takes the following arguments:

Table 10-112. **delete runtime** *arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`condition`	Condition when to delete it: `empty` or `always` **Attention:** The `always` option is dangerous.

Example

```
delete runtime mfp empty
```

This command is based on the "Runtime (DELETE)" on page 8-166 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]**

The **list farm-members** command takes the following arguments:

Table 10-113. **list farm-members** *arguments*

Argument	Description
`runtime-name`	Name of the runtime.

The **list farm-members** command takes the following options after the object.

Table 10-114. **`list farm-members`** *options*

Option	Description
`--xml`	Produces XML output instead of tabular output.

Example

```
list farm-members mfp
```

This command is based on the "Farm topology members (GET)" on page 8-114 REST service.

The `remove farm-member` command

The **`remove farm-member`** command removes a server from the list of farm members on which the specified runtime is deployed. Use this command when the server has become unavailable or disconnected.

Syntax: **remove farm-member [runtime-name] server-id**

The **`remove farm-member`** command takes the following arguments.

Table 10-115. **`remove farm-member`** *arguments*

Argument	Description
`runtime-name`	Name of the runtime.
`server-id`	Identifier of the server.

The **`remove farm-member`** command takes the following options after the object.

Table 10-116. **`remove farm-member`** *option*

Option	Description
`--force`	Force removal of a farm member, even if it is available and connected.

Example

```
remove farm-member mfp srvlx15
```

This command is based on the "Farm topology members (DELETE)" on page 8-116 REST service.

Federal standards support in IBM MobileFirst Platform Foundation

IBM MobileFirst Platform Foundation supports Federal Desktop Core Configuration (FDCC), and United States Government Configuration Baseline (USGCB) specifications. IBM MobileFirst Platform Foundation also supports the Federal Information Processing Standards (FIPS) 140-2, which is a security standard that is used to accredit cryptographic modules.

For more information about the Federal Desktop Core Configuration and United States Government Configuration Baseline, see FDCC and USGCB.

For more information about the Federal Information Processing Standards 140-2, see FIPS 140-2 support.

FDCC and USGCB support

The United States federal government mandates that federal agency desktops that run on Microsoft Windows platforms adopt Federal Desktop Core Configuration (FDCC) or the newer United States Government Configuration Baseline (USGCB) security settings.

IBM Worklight V5.0.6 was tested by using the USGCB and FDCC security settings via a self-certification process. Testing includes a reasonable level of testing to ensure that installation and core features function on this configuration.

References

For more information, 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. IBM MobileFirst Platform Foundation provides FIPS 140-2 support for Android, iOS, and Cordova apps.

FIPS 140-2 on the MobileFirst Server, and SSL communications with the MobileFirst Server

The IBM MobileFirst Platform Foundation server runs in an application server, such as the WebSphere Application Server. The WebSphere Application Server can be configured to enforce the use of FIPS 140-2 validated cryptographic modules for inbound and outbound Secure Socket Layer (SSL) connections. The cryptographic modules are also used for the cryptographic operations that are performed by the applications by using the Java™ Cryptography Extension (JCE). Since the 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 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.

Note: The cryptographic module instances that are used on the client are not necessarily FIPS 140-2 validated. For options to use FIPS 140-2 validated libraries on client devices, see "FIPS 140-2 on the MobileFirst client device for protection of data at rest in JSONStore and data in motion when using HTTPS communications" on page 10-76.
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 10-77 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. Protection of data in motion is provided by the use of HTTPS communication between the MobileFirst client and the MobileFirst Server.

On iOS devices, the FIPS 140-2 support is enabled by default for both data at rest and data in motion.

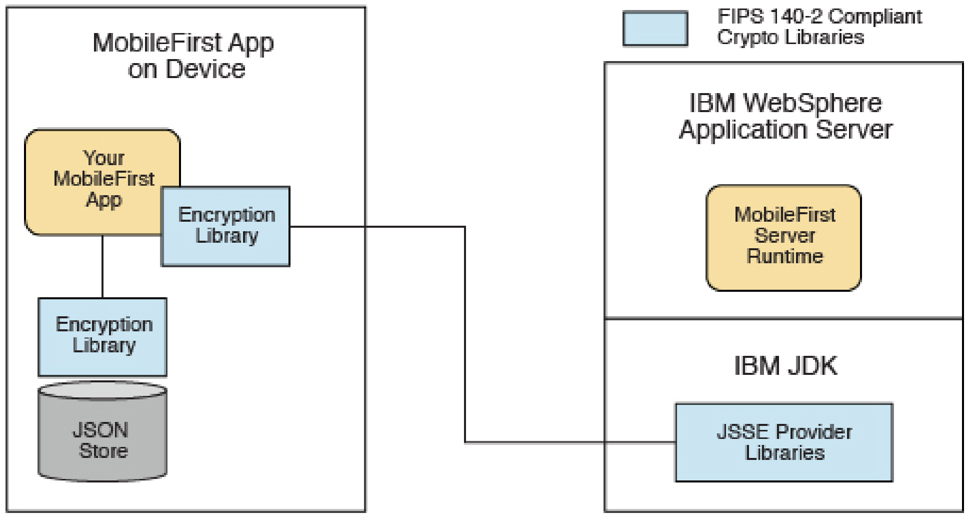
Android devices use non-FIPS 140-2 validated libraries by default. There is an option to use FIPS 140-2 validated libraries for the protection (encryption and decryption) of the local data that is stored by JSONStore and for the HTTPS communication to the MobileFirst Server. This support is achieved by using an OpenSSL library that achieved FIPS 140-2 validation (Certificate #1747). To enable this option in a MobileFirst client project, add the optional Android FIPS 140-2 plug-in.

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 only supported on the iOS and Android platforms.
  - On Android, this feature is only supported on devices or simulators that use the **`x86`** or **`armeabi`** architectures. It is not supported on Android using **`armv5`** or **`armv6`** architectures. The reason is because the OpenSSL library used did not obtain FIPS 140-2 validation for **`armv5`** or **`armv6`** on Android. FIPS 140-2 is not supported on 64-bit architecture even though the MobileFirst library does support 64-bit architecture. FIPS 140-2 can be run on 64-bit devices if the project includes only 32-bit native NDK libraries.
  - On iOS, it is supported on **`i386`**, **`x86_64`**, **`armv7`**, **`armv7s`**, and **`arm64`** architectures.
- This feature works with hybrid applications only (not with native applications).
- For native iOS, FIPS is enabled through the iOS FIPS libraries and is enabled by default. No action is required to enable FIPS 140-2.
- For HTTPS communications:
  - For Android devices, 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.
- IBM MobileFirst Platform Application Center client does not support the FIPS 140-2 feature.

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 Android or iOS environments.

Figure 10-4. 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 7-134.

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.

Enabling FIPS 140-2

To use the Federal Information Processing Standard (FIPS) 140-2 feature, you must first enable the FIPS 140-2 optional feature.

About this task

After the optional feature is enabled, it must then be configured as described in the *What to do next* section. After the FIPS 140-2 optional feature is enabled and configured, this feature applies both to HTTPS and JSONStore data encryption.

Note: FIPS 140-2 is only supported on Android and iOS. The iOS architectures that support FIPS 140-2 are **`i386`**, **`armv7`**, **`armv7s`**, **`x86_64`**, and **`arm64`**. The Android architectures that support FIPS 140-2 are **`x86`** and **`armeambi`**.

Note: On Android, FIPS 140-2 is not supported on 64-bit architecture even though the MobileFirst library does support 64-bit architecture. When you use FIPS 140-2 on a 64-bit device, you might see the following error:

```
java.lang.UnsatisfiedLinkError: dlopen failed: "..." is 32-bit instead of 64-bit
```

This error means that you have 64-bit native libraries in your Android project, and FIPS 140-2 does not currently work when you use these libraries. To confirm, go to `src/main/libs` or `src/main/jniLibs` under your Android project, and check whether you have the `x86_64` or `arm64-v8a` folders. If you do, delete these folders, and FIPS 140-2 can work again.

Note: The following considerations apply to enabling FIPS 140-2 on Cordova apps:

To use the FIPS 140-2 feature on the Android or iOS operating systems, complete the following steps:

Procedure

iOS only

1. For iOS, FIPS is enabled through the iOS FIPS libraries and is enabled by default. No action is required to enable FIPS 140-2.

Android only

2. Issue the following CLI command in your MobileFirst Android project:
   ```
   cordova plugin add cordova-plugin-mfp-fips
   ```

Results

FIPS 140-2 is enabled on your app. For the iOS operating system, the FIPS 140-2 compliance automatically applies to the JSONStore plugin when you install it.

What to do next

"Configure FIPS 140-2 mode for HTTPS and JSONStore encryption"

Configure FIPS 140-2 mode for HTTPS and JSONStore encryption

Learn about settings to configure FIPS 140-2 for encrypting data for HTTPS and JSONStore.

For iOS apps, FIPS 140-2 is enabled through the iOS FIPS libraries. It is enabled by default, so no action is required to enable or configure it.

The following code snippet is populated into a new IBM MobileFirst Platform Foundation application for the Android operating system in the `initOptions.js` file for configuring FIPS 140-2:

```
var wlInitOptions = {
  ...
  // # Enable FIPS 140-2 for data-in-motion (network) and data-at-rest (JSONStore) on Android.
  //   Requires the FIPS 140-2 optional feature to be enabled also.
  // enableFIPS : false
  ...
};
```

The default value of `enableFIPS` is false for the Android operating system. To enable FIPS 140-2 for both HTTPS and JSONStore data encryption, uncomment and set the option to `true`. After you set the value of `enableFIPS` to true, you should listen for the FIPS ready JavaScript event by creating a listening event similar to the following sample:

```
document.addEventListener('WL/FIPS/READY',
    this.onFipsReady, false);

onFipsReady: function() {
  // FIPS SDK is loaded and ready
}
```

After you set the value of the `enableFIPS` property, create an Android environment, and build the environment.

Note: You must install the FIPS Cordova plugin before you set the `enableFIPS` property value to *true*. Otherwise, a warning message is logged that states the `initOption` value is set, but the optional feature was not found. The FIPS 140-2 and JSONStore features are both optional on the Android operating system. FIPS 140-2 affects JSONStore data encryption only if the JSONStore optional feature is also enabled. If JSONStore is not enabled, then FIPS 140-2 does not affect JSONStore. In iOS, the FIPS 140-2 optional feature is not required for JSONStore FIPS 140-2 (data at rest) or HTTPS encryption (data in motion) because they are both handled by iOS. In Android, you must enable the FIPS 140-2 optional feature if you want to use JSONStore FIPS 140-2 or HTTPS encryption.

```
[WARN] FIPSHttp feature not found, but initOptions enables it on startup
```

After completing the procedure in "Enabling FIPS 140-2" on page 10-77 and this procedure in your Android environment, it automatically applies the FIPS 140-2 compliance to the JSONStore plug-in when you install it.

Configuring FIPS 140-2 for existing applications

You must modify applications that were created in earlier versions of IBM MobileFirst Platform Foundation to enable the FIPS 140-2 feature.

Before you begin

The FIPS 140-2 optional feature is not enabled by default on apps created for any versions of the Android operating system and on iOS apps in versions of IBM MobileFirst Platform Foundation before version 8.0. To enable the FIPS 140-2 optional feature for the Android operating system, see "Enabling FIPS 140-2" on page 10-77. After the optional feature is enabled, you can configure FIPS 140-2.

About this task

After you completed the steps that are described in "Enabling FIPS 140-2" on page 10-77, you must configure FIPS 140-2 by modifying the `initOptions.js` file to add the FIPS configuration property.

Note: For JSONStore FIPS 140-2 users The FIPS 140-2 feature, combined with the JSONStore feature, enables FIPS 140-2 support for JSONStore. This combination supersedes what was indicated in tutorial *JSONStore - Encrypting sensitive data with FIPS 140-2* that was available for IBM Worklight V6.0 or earlier. If you previously modified an application by following the instructions in this tutorial, delete and re-create its iPhone, iPad, and Android environments. Because any environment-specific changes that you previously made are lost when you delete an environment, make sure to back up any such changes before you delete any environment. After the environment is re-created, you can reapply those changes to the new environment.

Procedure

1. Add the following property to the `initOptions` object found in the `index.js` file.

   ```
   enableFIPS : true
   ```

2. Rebuild and deploy your app.

License tracking

License tracking is enabled by default in IBM MobileFirst Platform Foundation, 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 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 leaves the company, for example.

Setting the application license information

Learn how to set the application license information for the apps you register to MobileFirst Server.

About this task

License terms distinguish IBM MobileFirst Platform Foundation, IBM MobileFirst Platform Foundation Consumer, IBM MobileFirst Platform Foundation Enterprise, and IBM MobileFirst Platform Additional Brand Deployment. Set the license information of an application when you register it to a server so that license tracking reports generate the right license information. If your server is configured for token licensing, the license information is used to check out the right feature from the license server.

You set the Application Type and the Token License Type.

The possible values for Application Type are

`B2C`
: Use this application type if your application is licensed as IBM MobileFirst Platform Foundation Consumer.

`B2E`
: Use this application type if your application is licensed as IBM MobileFirst Platform Foundation Enterprise.

`UNDEFINED`
: Use this application type if you don't need to track compliance against the Addressable Device metric.

The possible values for Token License Type are

APPLICATION
: Use *APPLICATION* for most applications. This is the default.

ADDITIONAL_BRAND_DEPLOYMENT
: Use this *ADDITIONAL_BRAND_DEPLOYMENT* if your application is licensed as IBM MobileFirst Platform Additional Brand Deployment.

NON_PRODUCTION

Use *NON_PRODUCTION* while you are developing and testing the application on the production server. No token is checked out for applications that have a *NON_PRODUCTION* token license type.

Important: Using *NON_PRODUCTION* for a production app is a breach of the license terms.

Note: If your server is configured for token licensing and if you plan to register an application with Token License Type *ADDITIONAL_BRAND_DEPLOYMENT* or *NON_PRODUCTION*, set the application license information before you register the first version of the application. With **mfpadm** program, you can set the license information for an application before any version is registered. After the license information is set, the right number of tokens is checked out when you register the first version of the app. For more information about token validation, see "Token license validation" on page 10-83.

Procedure

Set the license type of your app

- To set the license type with IBM MobileFirst Platform Operations Console
  1. Select your application
  2. Select **Settings**
  3. Set the Application Type and the Token License Type
  4. Click **Save**
- To set the license type with the **mfpadm** program,
  1. Use **mfpadm app <appname> set license-config <application-type> <token license type>**

     The following example sets the license information B2E / APPLICATION to the application named *my.test.application*

     ```
     prompt> echo password:admin > password.txt
     prompt> mfpadm --url https://localhost:9443/mfpadmin --secure false --user admin \
             --passwordfile password.txt \
             app mfp my.test.application ios 0.0.1 set license-config B2E APPLICATION
     prompt> rm password.txt
     ```

  For more information about **mfpadm**, see "Administering MobileFirst applications through the command line" on page 10-47.

License Tracking report

IBM MobileFirst Platform Foundation provides a license tracking report for the Client Device metric, the Addressable Device metric, and the Application metric. The report also provides historical data.

The License Tracking report shows the following data:

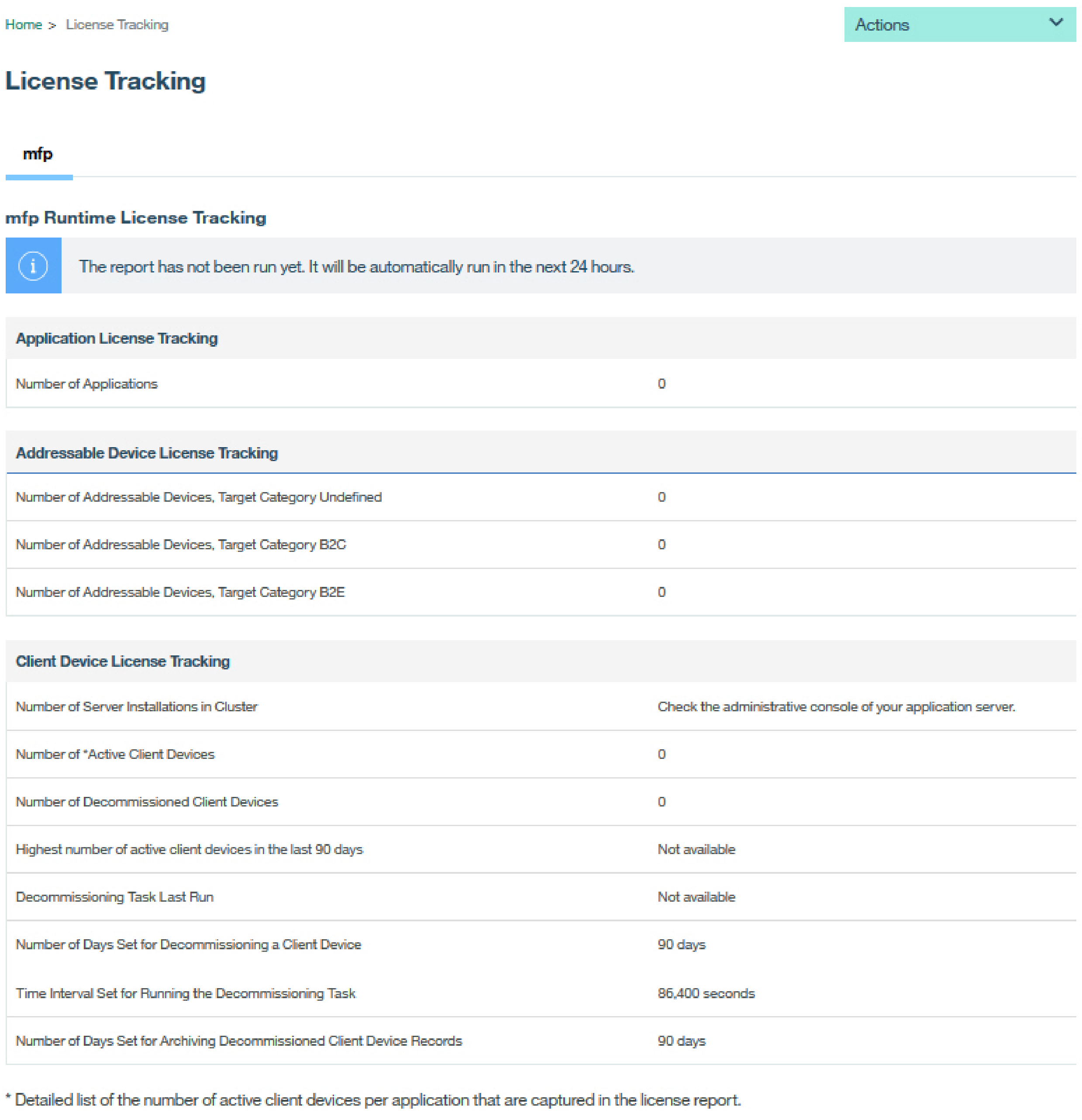
- The number of applications deployed in the IBM MobileFirst Platform Server.
- The number of addressable devices in the current calendar month.
- The number of client devices, both active and decommissioned.
- The highest number of client devices reported over the last *n* days, where *n* is the number of days of inactivity after which a client device is decommissioned.

You might want to analyze data further. For this purpose, you can download a CSV file that includes the license reports as well as a historical listing of license metrics.

To access the License Tracking report,

1. Open IBM MobileFirst Platform Operations Console.
2. Click the **Hello, <your Name>** menu.
3. Select **Licenses**.
4. To obtain a CSV file from the License Tracking report, click **Actions/Download report**.

,



License Tracking

mfp Runtime License Tracking

The report has not been run yet. It will be automatically run in the next 24 hours.

Application License Tracking	
Number of Applications	0

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 Installations in Cluster	Check the administrative console of your application server.
Number of *Active Client Devices	0
Number of Decommissioned Client Devices	0
Highest number of active client devices in the last 90 days	Not available
Decommissioning Task Last Run	Not available
Number of Days Set for Decommissioning a Client Device	90 days
Time Interval Set for Running the Decommissioning Task	86,400 seconds
Number of Days Set for Archiving Decommissioned Client Device Records	90 days

* Detailed list of the number of active client devices per application that are captured in the license report.

Figure 10-5. License tracking information for applications, devices, and decommissioning

For more information about configuring licenses tracking, see "Configuring license tracking for client device and addressable device" on page 6-196.

Token license validation

If you install and configure IBM MobileFirst Platform Server for token licensing, the server validates licenses in various scenarios. If your configuration is not correct, the license is not validated at application registration or deletion.

Validation scenarios

Licenses are validated in various scenarios:

On application registration
Application registration fails if not enough tokens are available for the token license type of your application.

Tip: You can set the token license type before you register the first version of your app. For more information, see "Setting the application license information" on page 10-80.

Licenses are checked only once per application. If you register a new platform for the same application, or if you register a new version for an existing application and platform, no new token is claimed.

On Token License Type change
When you change the Token License Type for an application, the tokens for the application are released and then taken back for the new license type.

On application deletion
Licenses are checked in when the last version of an application is deleted.

At server start
The license is checked out for every registered application. The server deactivates applications if not enough tokens are available for all applications.

Important: The server does not reactivate the applications automatically. After you increase the number of available tokens, you must reactivate the applications manually. For more information about disabling and enabling applications, see "Remotely disabling application access to protected resources" on page 10-17.

On license expiration
After a certain amount of time, the licenses expire and must be checked out again. The server deactivates applications if not enough tokens are available for all applications.

Important: The server does not reactivate the applications automatically. After you augment the number of available tokens, you must reactivate the applications manually. For more information about disabling and enabling applications, see "Remotely disabling application access to protected resources" on page 10-17.

At server shutdown
The license is checked in for every deployed application, during a server shutdown. The tokens are released only when the last server of a cluster of farm is shut down.

Causes of license validation failure

License validation might fail when the application is registered or deleted, in the following cases:

- The Rational Common Licensing native library is not installed and configured.
- The administration service is not configured for token licensing. For more information, see “Installing and configuring for token licensing” on page 6-150.
- Rational License Key Server is not accessible.
- Sufficient tokens are not available.
- The license expired.

IBM Rational License Key Server feature name used by IBM MobileFirst Platform Foundation

Depending on the token license type of an application, the following features are used.

Token License Type	Feature name
APPLICATION	`ibmmfpfa`
ADDITIONAL_BRAND_DEPLOYMENT	`ibmmfpabd`
NON_PRODUCTION	(no feature)

For more information about setting the token license type, see “Setting the application license information” on page 10-80.

Integration with IBM License Metric Tool

The IBM License Metric Tool allows you to evaluate your compliance with your IBM license.

If you have not installed a version of IBM License Metric Tool that supports IBM Software License Metric Tag or SWID (software identification) files, you can review the license usage with the License Tracking reports in MobileFirst Operations Console. For more information, see “License Tracking report” on page 10-81.

About PVU-based licensing using SWID files

If you have purchased IBM MobileFirst Platform Foundation Extension V8.0.0 offering, it is licensed under the Processor Value Unit (PVU) metric.

The PVU calculation is based on IBM License Metric Tool's support for **ISO/IEC 19970-2** and **SWID** files. The SWID files are written to the server when the IBM Installation Manager installls MobileFirst or MobileFirst Analytics Server. When the IBM License Metric Tool discovers an invalid SWID file for a product according to the current catalog, a warning sign is displayed on the Software Catalog widget. For more information on how the IBM License Metric Tool works with SWID files, see https://www.ibm.com/support/knowledgecenter/SS8JFY_9.2.0/com.ibm.lmt.doc/Inventory/overview/c_iso_tags.html.

The number of Application Center installations is not limited by PVU-based licensing.

The PVU license for Foundation Extension can only be purchased together with these product licenses: IBM WebSphere Application Server Network Deployment, IBM API Connect Professional, or IBM API Connect Enterprise. IBM Installation Manager adds or updates the SWID file to be used by the License Metric Tool. For more information on IBM MobileFirst Foundation Extension, see https://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=AN&subtype=CA&htmlfid=897/ENUS216-367&appname=USN.

For more information on PVU licensing see https://www.ibm.com/support/knowledgecenter/SS8JFY_9.2.0/com.ibm.lmt.doc/Inventory/overview/c_processor_value_unit_licenses.html.

SLMT tags

IBM MobileFirst Platform Foundation 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.

Each instance of a running MobileFirst runtime environment generates an IBM Software License Metric Tag file. The metrics monitored are `CLIENT_DEVICE`, `ADDRESSABLE_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 license tracking for client device and addressable device" on page 6-196.
- 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 license tracking for client device and addressable device" on page 6-196.

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".

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).

About the `ADDRESSABLE_DEVICE` metric

The `ADDRESSABLE_DEVICE` metric has the following subtype:

- Application: `<applicationName>`, Category: `<application type>`

The application type is `B2C`, `B2E`, or `UNDEFINED`. To define the application type of an application, see “Setting the application license information” on page 10-80.

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 license tracking was disabled, no addressable report is generated.

For more information about configuring license tracking using metrics, see

- “Configuring license tracking for client device and addressable device” on page 6-196
- “Configuring IBM License Metric Tool log files” on page 6-197

Analytics and Logger

IBM MobileFirst Analytics gives a rich view into both your mobile landscape and server infrastructure. Included are default reports of user retention, crash reports, device type and operating system breakdowns, custom data and custom charts, network usage, push notification results, in-app behavior, debug log collection, and beyond.

IBM MobileFirst Platform Server comes pre-instrumented with network infrastructure reporting. When both the client and server are reporting their network usage, the data is aggregated so you can attribute poor performance to the network, the server, or the back-end systems.

Two client classes work together to send raw data to the server: the web logger class and the web analytics class. The logger functions as a standard logger. In addition, you can control which logger data is accessed and used by analytics by defining filters both on the client side and on the MobileFirst Analytics Server.

You choose the verbosity and data retention policy of the reported events. Set conditional alerts. Build custom charts. Engage with new data.

Supported platforms

Analytics is available in iOS and Android. For Cordova, it is available for Android and iOS platforms. It is not available for the Windows API.

New: From V8.0.0, analytics is available also in the new web API. For more information on this new API, see "Developing web applications" on page 7-73. To access analytics for your web apps, use the classes and methods documented here and in JavaScript web analytics client-side API, and not WL.Analytics and WL.Logger.

Major features

Learn about the major features that are provided with MobileFirst Analytics and Logger.

Built-in Analytics

When you use the MobileFirst client SDK together with the MobileFirst Server, analytics data automatically gets collected for any request that your app makes to the MobileFirst Server. Basic device metadata gets collected and reported to the MobileFirst Analytics Server.

App Analytics

You can view App Session charts and App Usage charts to find out which app is being used most by your users.

Custom Analytics

You can have your app send custom data and create custom reports on your custom data.

Custom Charts

You can create custom charts for the custom data that you collect or many of the pre-defined analytics data types.

Crash Capture

MobileFirst client SDKs can capture crashes on Android and iOS, and uncaught exceptions in cross-platform and web applications.

Alerts

You can create threshold and trend alerts in the MobileFirst Analytics Console.

Debug Log Capture

Log capture gives developers the ability to easily capture raw debug logs from applications that run on user devices that are inaccessible to your development team.

REST API

You can view the “REST API for MobileFirst Analytics and Logger” on page 8-270.

MobileFirst Analytics Server installation guide

MobileFirst Analytics Server is implemented and shipped as a set of two Java EE standard web application archive (WAR) files, or one enterprise application archive (EAR) file. Therefore, it can be installed in one of the following supported application servers: WebSphere Application Server, WebSphere Application Server Liberty, or Apache Tomcat (WAR files only).

System requirements

MobileFirst Analytics Server uses an embedded Elasticsearch library for the data store and cluster management. Because it intends to be a highly performant in-memory search and query engine, requiring fast disk I/O, you must follow some production system requirements. In general, you are most likely to run out of memory and disk (or discover that disk I/O is your performance bottleneck) before CPU becomes a problem. In a clustered environment, you want a fast, reliable, co-located cluster of nodes.

Operating system

- CentOS/RHEL 6.x/7.x
- Oracle Enterprise Linux 6/7 with RHEL Kernel only
- Ubuntu 12.04/14.04
- SLES 11/12
- OpenSuSE 13.2
- Windows Server 2012/R2
- Debian 7

JVM

- Oracle JVM 1.7u55+
- Oracle JVM 1.8u20+

- IcedTea OpenJDK 1.7.0.55+

Hardware

- RAM: More RAM is better, but no more than 64 GB per node. 32 GB and 16 GB are also acceptable. Less than 8 GB requires many small nodes in the cluster, and 64 GB is wasteful and problematic due to the way Java uses memory for pointers.
- Disk: Use SSDs when possible, or fast spinning traditional disks in RAID 0 configuration if SSDs are not possible.
- CPU: CPU tends not to be the performance bottleneck. Use systems with 2 to 8 cores.
- Network: When you cross into the need to scale out horizontally, you need a fast, reliable, data center with 1 GbE to 10 GbE supported speeds.

Hardware configuration

- Give your JVM half of the available RAM, but do not cross 32 GB
  - Setting the `ES_HEAP_SIZE` environment variable to `32g`.
  - Setting the JVM flags by using `-Xmx32g -Xms32g`.
- Turn off disk swap. Allowing the operating system to swap heap on and off disk significantly degrades performance.
  - Temporarily: `sudo swapoff -a`
  - Permanently: Edit `/etc/fstab` according to the operating system documentation.
  - If neither option is possible, set the Elasticsearch option `bootstrap.mlockall: true` (this value is the default in the embedded Elasticsearch instance).
- Increase the allowed open file descriptors.
  - Linux typically limits a per-process number of open file descriptors to a small 1024.
  - Consult your operating system documentation for how to permanently increase this value to something much larger, like 64,000.
- Elasticsearch also uses a mix of NioFS and MMapFS for the various files. Increase the maximum map count so plenty of virtual memory is available for mmapped files.
  - Temporarily: `sysctl -w vm.max_map_count=262144`
  - Permanently: Modify the `vm.max_map_count` setting in your `/etc/sysctl.conf`.
- If you use BSDs and Linux, ensure that your operating system I/O scheduler is set to `deadline` or `noop`, not `cfq`.

Capacity considerations

Capacity is the single-most common question. How much RAM do you need? How much disk space? How many nodes? The answer is always: it depends.

IBM MobileFirst Analytics gives you the opportunity to collect many heterogeneous event types, including raw client SDK debug logs, server-reported network events, custom data, and much more. It is a big data system with big data system requirements.

The type and amount of data that you choose to collect, and how long you choose to keep it, has a dramatic impact on your storage requirements and overall performance. As an example, consider the following questions.

- Are raw debug client logs useful after a month?

- Are you using the **Alerts** feature in MobileFirst Analytics? If so, are you querying on events that occurred in the last few minutes or over a longer range?
- Are you using custom charts? If so, are you creating these charts for built-in data or custom instrumented key/value pairs? How long do you keep the data?

The built-in charts on the MobileFirst Analytics Console are rendered by querying data that the MobileFirst Analytics Server already summarized and optimized specifically for the fastest possible console user experience. Because it is pre-summarized and optimized for the built-in charts, it is not suitable for use in alerts or custom charts where the console user defines the queries.

When you query raw documents, apply filters, perform aggregations, and ask the underlying query engine to calculate averages and percentages, the query performance necessarily suffers. It is this use case that requires careful capacity considerations. After your query performance suffers, it is time to decide whether you really must keep old data for real-time console visibility or purge it from the MobileFirst Analytics Server. Is real-time console visibility truly useful for data from four months ago?

Indicies, Shards, and Nodes

The underlying data store is Elasticsearch. You must know a bit about indices, shards and nodes, and how the configuration affects performance. Roughly, you can think of an index as a logical unit of data. An index is mapped one-to-many to shards where the configuration key is `shards`. For more information, see "Configuration guide" on page 11-14. The MobileFirst Analytics Server creates a separate index per document type. If your configuration does not discard any document types, you have a number of indices that are created that is equivalent to the number of document types that are offered by the MobileFirst Analytics Server.

If you configure the `shards` to 1, each index only ever has one primary shard to which data is written. If you set `shards` to 10, each index can balance to 10 shards. However, more shards have a performance cost when you have only one node. That one node is now balancing each index to 10 shards on the same physical disk. Only set `shards` to 10 if you plan to immediately (or nearly immediately) scale up to 10 physical nodes in the cluster.

The same principle applies to `replicas`. Only set `replicas` to something greater than 0 if you intend to immediately (or nearly immediately) scale up to the number of nodes to match the math.

For example, if you set `shards` to 4 and `replicas` to 2, you can scale to 8 nodes, which is 4 * 2.

Installing MobileFirst Analytics on WebSphere Application Server Liberty

You can install MobileFirst Analytics on WebSphere Application Server Liberty.

Before you begin

Ensure that you already have the MobileFirst Analytics EAR file. For more information on the installation artifacts, see "Installing MobileFirst Server to an application server" on page 6-100. The `analytics.ear` file is found in the *<mf_server_install_dir>*`\analytics` folder. For more information about how to download and install WebSphere Application Server Liberty, see the About WebSphere Liberty article on IBM developerWorks.

Procedure

1. Create a server by running the following command in your `./wlp/bin` folder.

   ```
   ./server create <serverName>
   ```

2. Install the following features by running the following command in your `./bin` folder.

   ```
   ./featureManager install jsp-2.2 ssl-1.0 appSecurity-1.0 localConnector-1.0
   ```

3. Add the `analytics.ear` file to the `./usr/servers/<serverName>/apps` folder of your Liberty Server.
4. Replace the contents of the `<featureManager>` tag of the `./usr/servers/<serverName>/server.xml` file with the following content.

   ```
   <featureManager>
     <feature>jsp-2.2</feature>
     <feature>ssl-1.0</feature>
     <feature>appSecurity-1.0</feature>
     <feature>localConnector-1.0</feature>
   </featureManager>
   ```

5. Configure `analytics.ear` as an application with role-based security in the `server.xml` file. The following example creates a basic hardcoded user registry, and assigns a user to each of the different analytics roles.

   ```
   <application location="analytics.ear" name="analytics-ear" type="ear">
     <application-bnd>
       <security-role name="analytics_administrator">
         <user name="admin"/>
       </security-role>
       <security-role name="analytics_infrastructure">
         <user name="infrastructure"/>
       </security-role>
       <security-role name="analytics_support">
         <user name="support"/>
       </security-role>
       <security-role name="analytics_developer">
         <user name="developer"/>
       </security-role>
       <security-role name="analytics_business">
         <user name="business"/>
       </security-role>
     </application-bnd>
   </application>

   <basicRegistry id="worklight" realm="worklightRealm">
     <user name="business" password="demo"/>
     <user name="developer" password="demo"/>
     <user name="support" password="demo"/>
     <user name="infrastructure" password="demo"/>
     <user name="admin" password="admin"/>
   </basicRegistry>
   ```

   For more information about how to configure other user registry types, such as LDAP, see the Configuring a user registry for Liberty topic in the WebSphere Application Server product documentation.
6. Start the Liberty Server by running the following command inside your `bin` folder.

   ```
   ./server start <serverName>
   ```

7. Go to the MobileFirst Analytics Console.

   ```
   http://localhost:9080/analytics/console
   ```

What to do next

For more information about administering WebSphere Application Server Liberty, see the Administering Liberty from the command line topic in the WebSphere Application Server product documentation.

Installing MobileFirst Analytics on Tomcat

You can install MobileFirst Analytics on Apache Tomcat.

Before you begin

Ensure that you already have the MobileFirst Analytics WAR files. For more information on the installation artifacts, see "Installing MobileFirst Server to an application server" on page 6-100. The `analytics-ui.war` and `analytics-service.war` files are found in the `<mf_server_install_dir>\analytics` folder. For more information about how to download and install Tomcat, see Apache Tomcat. Ensure that you download the version that supports Java 7 or higher. For more information about which version of Tomcat supports Java 7, see Apache Tomcat Versions.

Procedure

1. Add `analytics-service.war` and the `analytics-ui.war` files to the Tomcat `webapps` folder.
2. Uncomment the following section in the `conf/server.xml` file, which is present, but commented out, in a freshly downloaded Tomcat archive.
   ```
   <Valve className ="org.apache.catalina.authenticator.SingleSignOn"/>
   ```
3. Declare the two war files in the `conf/server.xml` file, and define a user registry.
   ```
   <Context docBase ="analytics-service" path ="/analytics-service"></Context>
   <Context docBase ="analytics" path ="/analytics"></Context>
   <Realm className ="org.apache.catalina.realm.MemoryRealm"/>
   ```

   The `MemoryRealm` recognizes the users that are defined in the `conf/tomcat-users.xml` file. For more information about other choices, see Apache Tomcat Realm Configuration HOW-TO.
4. Add the following sections to the `conf/tomcat-users.xml` file to configure a `MemoryRealm`.
   a. Add the security roles.
      ```
      <role rolename="analytics_administrator"/>
      <role rolename="analytics_infrastructure"/>
      <role rolename="analytics_support"/>
      <role rolename="analytics_developer"/>
      <role rolename="analytics_business"/>
      ```
   b. Add a few users with the roles you want.
      ```
      <user name="admin" password="admin" roles="analytics_administrator"/>
      <user name="support" password="demo" roles="analytics_support"/>
      <user name="business" password="demo" roles="analytics_business"/>
      <user name="developer" password="demo" roles="analytics_developer"/>
      <user name="infrastructure" password="demo" roles="analytics_infrastructure"/>
      ```
5. Start your Tomcat Server and go to the MobileFirst Analytics Console.
   ```
   http://localhost:8080/analytics/console
   ```

   For more information about how to start the Tomcat Server, see the official Tomcat site. For example, Apache Tomcat 7, for Tomcat 7.0.

Installing MobileFirst Analytics on WebSphere Application Server

You can install MobileFirst Analytics on WebSphere Application Server.

Before you begin

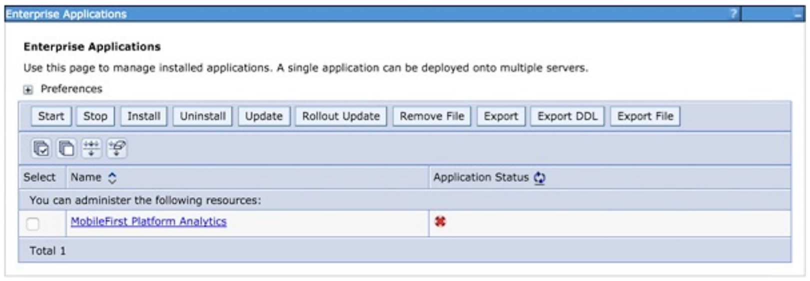
For more information on initial installation steps for acquiring the installation artificats (JAR and EAR files), see "Installing MobileFirst Server to an application server" on page 6-100. The `analytics.ear`, `analytics-ui.war`, and `analytics-service.war` files are found in the `<mf_server_install_dir>\analytics` folder.

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 - 7 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 how to install an EAR file on WebSphere Application Server, see the Installing enterprise application files with the console topic in the WebSphere Application Server product documentation.
2. Select the **MobileFirst Platform Analytics** application from the **Enterprise Applications** list.

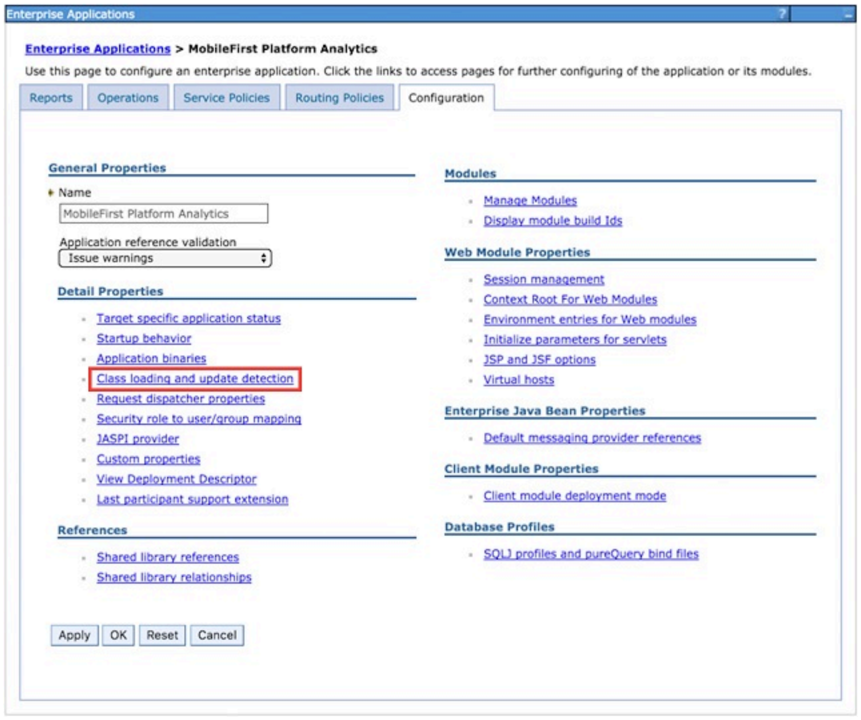


Enterprise Applications

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

Select	Name	Application Status
You can administer the following resources:		
☐	MobileFirst Platform Analytics	✖
Total 1		

3. Click **Class loading and update detection**.


Enterprise Applications

Enterprise Applications > **MobileFirst Platform Analytics**

Use this page to configure an enterprise application. Click the links to access pages for further configuring of the application or its modules.

Reports | Operations | Service Policies | Routing Policies | Configuration

General Properties

* Name: MobileFirst Platform 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
- 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

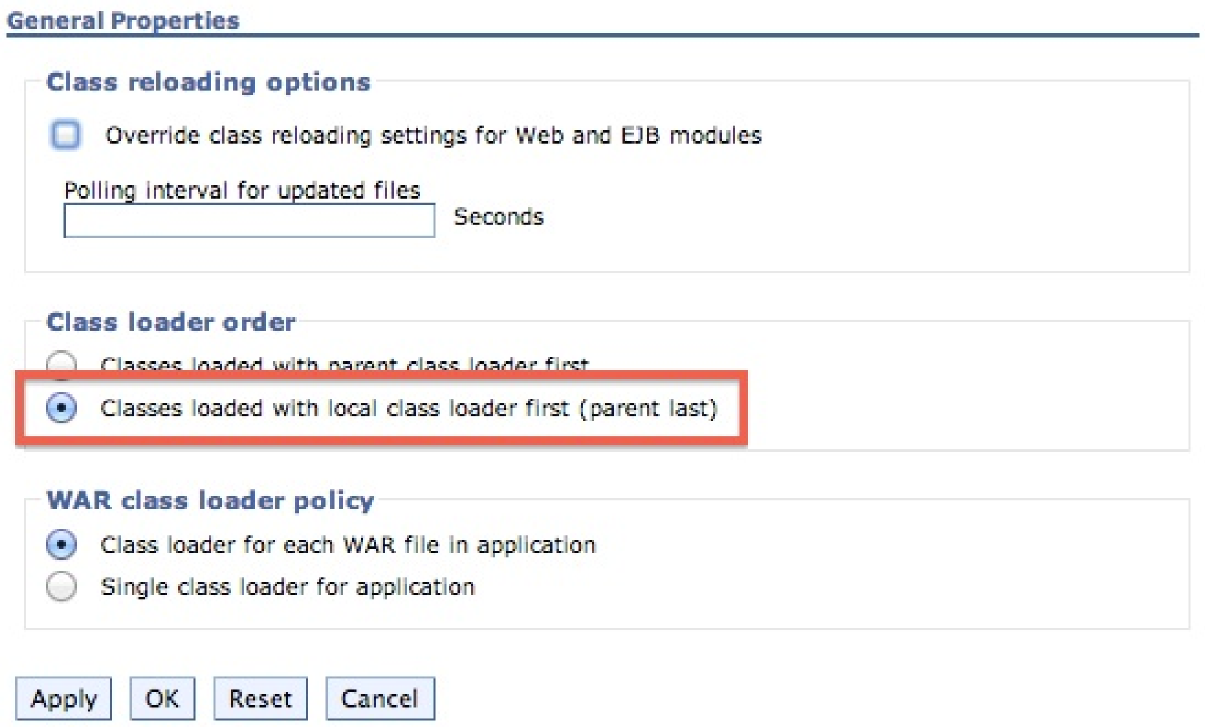
- Client module deployment mode

Database Profiles

- SQLJ profiles and pureQuery bind files

Apply | OK | Reset | Cancel

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)]

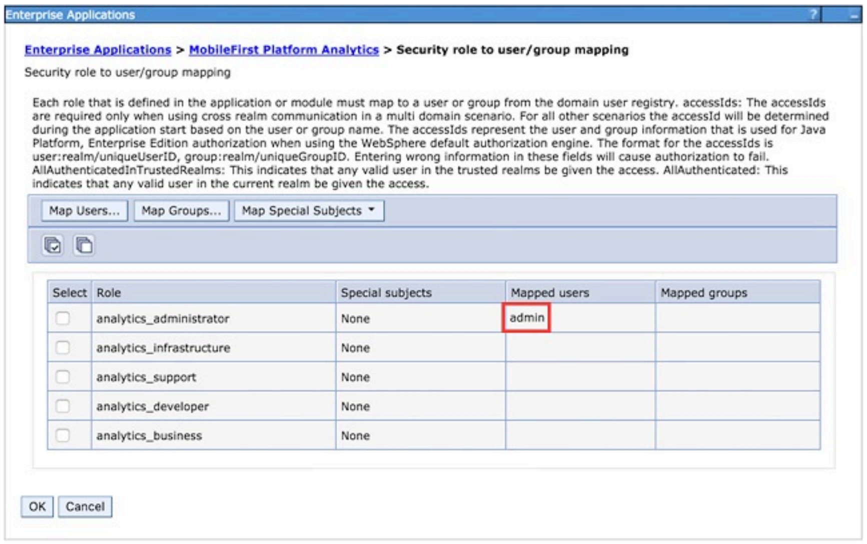
WAR class loader policy

[✓ Class loader for each WAR file in application]

○ Single class loader for application

Apply | OK | Reset | Cancel

5. Click **Security role to user/group mapping** to map the admin user.



Enterprise Applications

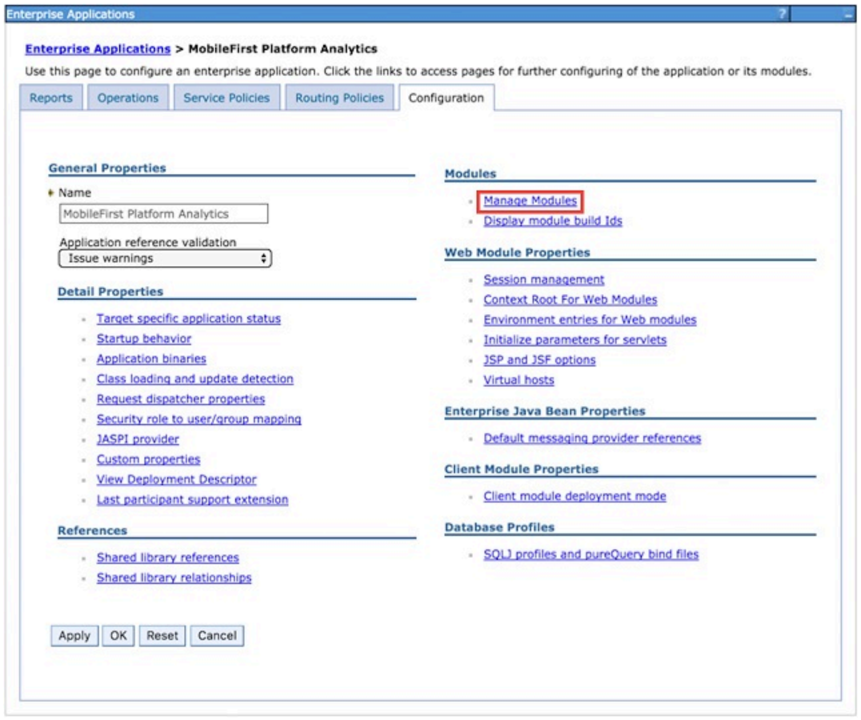
Enterprise Applications > MobileFirst Platform Analytics > **Security role to user/group mapping**

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
☐	analytics_administrator	None	admin	
☐	analytics_infrastructure	None		
☐	analytics_support	None		
☐	analytics_developer	None		
☐	analytics_business	None		

6. Click **Manage Modules**.



Enterprise Applications

Enterprise Applications > **MobileFirst Platform 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: MobileFirst Platform 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
- 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

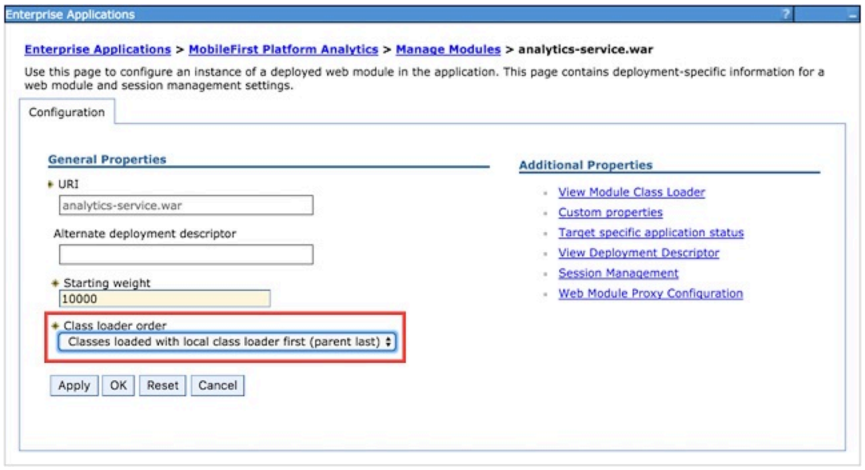
Client Module Properties

- Client module deployment mode

Database Profiles

- SQLJ profiles and pureQuery bind files

7. Select the **analytics** module and change the class loader order to **parent last**.



Enterprise Applications > MobileFirst Platform Analytics > Manage Modules > 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.

Configuration

General Properties

URI: analytics-service.war

Alternate deployment descriptor:

Starting weight: 10000

Class loader order: Classes loaded with local class loader first (parent last)

8. Enable **Administrative security** and **application security** in the WebSphere Application Server administration console:
   a. Log in to the WebSphere Application Server administration console.
   b. In the **Security** > **Global Security** menu, ensure that **Enable administrative security** and **Enable application security** are both selected. **Note:** **Application security** can be selected only after **administrative security** is enabled.
   a. Click **OK** and save changes.
9. Start the **MobileFirst Platform Analytics** application and go to the link in the browser.

   `http://<hostname>:<port>/analytics/console`

Results

The Analytics EAR file is now ready to accept incoming analytics data.

Installing MobileFirst Analytics with Ant tasks

Learn how to use Ant tasks to deploy MobileFirst Analytics to your application server.

Before you begin

Ensure that you have the necessary WAR and configuration files: `analytics-ui.war` and `analytics-service.war`. For more information on the installation artifacts, see "Installing MobileFirst Server to an application server" on page 6-100. The `analytics-ui.war` and `analytics-service.war` files are found in the `MobileFirst_Platform_Server\analytics`.

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>/MobileFirstServer/mfp-ant-deployer.jar` to that computer.

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

Procedure

1. Edit the Ant script that you use later to deploy MobileFirst Analytics WAR files.
   a. Review the sample configuration files in "Sample configuration files for MobileFirst Analytics" on page 6-319.
   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 sets the `masternodes` JNDI property to that value.
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 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.

   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 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

- "Ant tasks for installation of MobileFirst Analytics" on page 6-309

Installing MobileFirst Analytics Server V8.0.0 on servers running previous versions

Although there is no option to upgrade previous versions of the MobileFirst Analytics Server, when you install MobileFirst Analytics Server V8.0.0 on a server that hosted a previous version, some properties and analytics data need to be migrated.

For servers previously running earlier of versions of MobileFirst Analytics Server update the analytics data and the JNDI properties.

Migration of server properties used by previous versions of MobileFirst Analytics Server

If you install MobileFirst Analytics Server V8.0.0 on a server that was previously running an earlier version of MobileFirst Analytics Server, you must update the values of the JNDI properties on the hosting server.

Some event types were changed between earlier versions of MobileFirst Analytics Server and V8.0.0. Because of this change, any JNDI properties that were previously configured in your server configuration file must be converted to the new event type.

The following table shows the mapping between old event types and new event types. Some event types did not change.

Table 11-1. MobileFirst Server properties

Old event type	New event type
`AlertDefinition`	`AlertDefinition`
`AlertNotification`	`AlertNotification`
`AlertRunnerNode`	`AlertRunnerNode`
`AnalyticsConfiguration`	`AnalyticsConfiguration`
`CustomCharts`	`CustomChart`
`CustomData`	`CustomData`
`Devices`	`Device`
`MfpAppLogs`	`AppLog`
`MfpAppPushAction`	`AppPushAction`
`MfpAppSession`	`AppSession`
`ServerLogs`	`ServerLog`
`ServerNetworkTransactions`	`NetworkTransaction`
`ServerPushNotifications`	`PushNotification`
`ServerPushSubscriptions`	`PushSubscription`
`Users`	`User`
`inboundRequestURL`	`resourceURL`
`mfpAppName`	`appName`
`mfpAppVersion`	`appVersion`

Analytics data migration

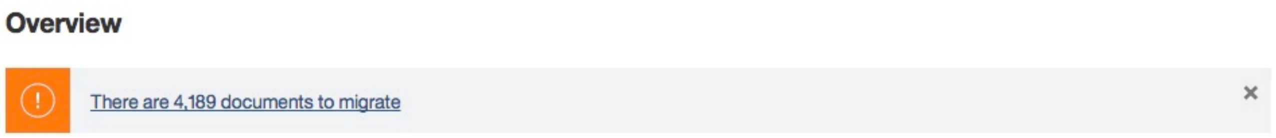
Learn about migrating data in the IBM MobileFirst Analytics.

The internals of the MobileFirst 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.

When you first view the MobileFirst Analytics Console after you upgrade to V8.0.0, no statistics are rendered in the MobileFirst Analytics Console. Your data is not lost, but it must be migrated to the new data format.

An alert is displayed on every page of the MobileFirst Analytics Console that reminds you that documents must be migrated. The alert text includes a link to the **Migration** page.

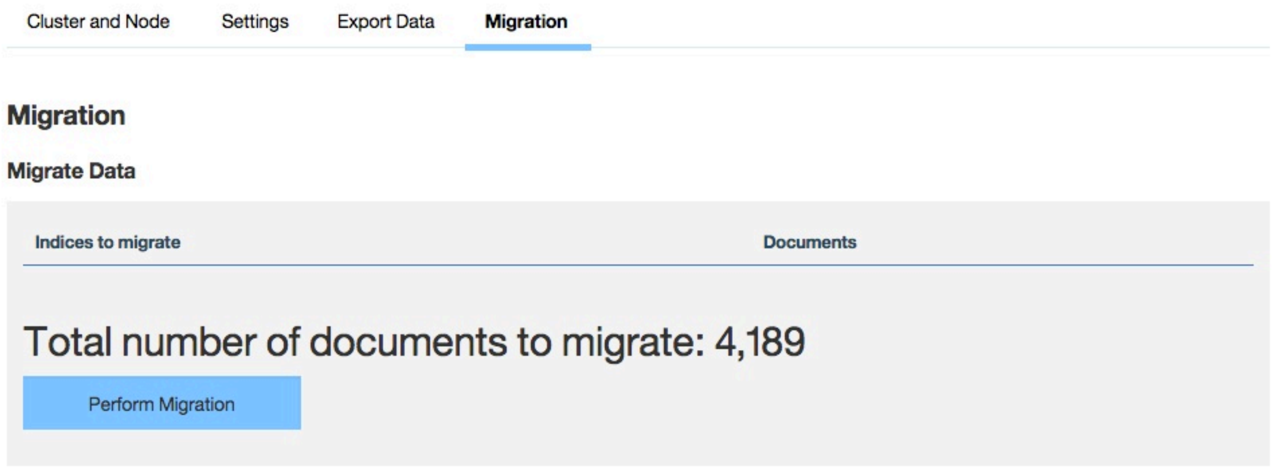
The following image shows a sample alert from the **Overview** page of the **Dashboard** section:



Migration page

You can access the **Migration** page from the wrench icon in the MobileFirst Analytics Console. From the **Migration** page, you can see how many documents must be migrated, and which indices they are stored on. Only one action is available: **Perform Migration**.

The following image shows the **Migration** page when you have documents that must be migrated:



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 can take 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, so they are not rendered in the MobileFirst Analytics Console.

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.

Configuration guide

Some configuration for the MobileFirst Analytics Server is required. Some of the configuration parameters apply to a single node, and some apply to the whole cluster, as indicated.

Properties

For a complete list of configuration properties and how to set them in your application server, see "Configuration properties" on page 11-15.

- The `discovery.zen.minimum_master_nodes` property must be set to `ceil((<number of master-eligible nodes in the cluster> / 2) + 1)` to avoid split-brain syndrome.
  - Elasticsearch nodes in a cluster that are master-eligible must establish a quorum to decide which master-eligible node is the master.
  - If you add a master eligible node to the cluster, the number of master-eligible nodes changes, and thus the setting must change. You must modify the setting if you introduce new master-eligible nodes to the cluster. For more information about how to manage your cluster, see "Cluster management and Elasticsearch" on page 11-19.
- Give your cluster a name by setting the `clustername` property in all of your nodes.
  - Name the cluster to prevent a developer's instance of Elasticsearch from accidentally joining a cluster that is using a default name.
- Give each node a name by setting the `nodename` property in each node.
  - By default, Elasticsearch names each node after a random Marvel character, and the node name is different on every node restart.
- Explicitly declare the file system path to the data directory by setting the `datapath` property in each node.
- Explicitly declare the dedicated master nodes by setting the `masternodes` property in each node.

Cluster Recovery Settings

After you scaled out to a multi-node cluster, you might find that an occasional full cluster restart is necessary. When a full cluster restart is required, you must consider the recovery settings. If the cluster has 10 nodes, and as the cluster is brought up, one node at a time, the master node assumes that it needs to start balancing data immediately upon the arrival of each node into the cluster. If the master is allowed to behave this way, much unnecessary rebalancing is required. You must configure the cluster settings to wait for a minimum number of nodes to join the cluster before the master is allowed to start instructing the nodes to rebalance. It can reduce cluster restarts from hours down to minutes.

- The `gateway.recover_after_nodes` property must be set to your preference to prevent Elasticsearch from starting a rebalance until the specified number of nodes in the cluster are up and joined. If your cluster has 10 nodes, a value of 8 for the `gateway.recover_after_nodes` property might be a reasonable setting.
- The `gateway.expected_nodes` property must be set to the number of nodes that you expect to be in the cluster. In this example, the value for the `gateway.expected_nodes` property is 10.

- The `gateway.recover_after_time` property must be set to instruct the master to wait to send rebalanced instructions until after the set time elapsed from the start of the master node.

The combination of the previous settings means that Elasticsearch waits for the value of `gateway.recover_after_nodes` nodes to be present. Then, it begins recovering after the value of `gateway.recover_after_time` minutes or after the value of `gateway.expected_nodes` nodes joined the cluster, whichever comes first.

What not to do

- Do not ignore your production cluster.
  - Clusters need monitoring and nurturing. Many good Elasticsearch monitoring tools are available that are dedicated to the task.
- Do not use network-attached storage (NAS) for your `datapath` setting. NAS introduces more latency, and a single point of failure. Always use the local hosts disks.
- Avoid clusters that span data centers and definitely avoid clusters that span large geographic distances. The latency between nodes is a severe performance bottleneck.
- Roll your own cluster configuration management solution. Many good configuration management solutions, such as Puppet, Chef, and Ansible, are available.

Configuration properties

The MobileFirst Analytics Server can start successfully without any additional configuration.

Configuration is done through JNDI properties on both the MobileFirst Server and the MobileFirst Analytics Server. Additionally, the MobileFirst Analytics Server supports the use of environment variables to control configuration. Environment variables take precedence over JNDI properties.

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.

To set a JNDI property on WebSphere Application Server Liberty, add a tag to the `server.xml` file as follows.

```
<jndiEntry jndiName="{{PROPERTY NAME}}" value="{{PROPERTY VALUE}}" />
```

To set a JNDI property on Tomcat, add a tag to the `context.xml` file as follows.

```
<Environment name="{{PROPERTY NAME}}" value="{{PROPERTY VALUE}}" type="java.lang.String" override="false" />
```

The JNDI properties on WebSphere Application Server are available as environment variables.

1. In the WebSphere Application Server console, select **Applications** > **Application Types** > **WebSphere Enterprise applications**.
2. Select the MobileFirst Administration Service application.
3. In **Web Module Properties**, click **Environment entries for Web Modules** to display the JNDI properties.

MobileFirst Server

The following table shows the properties that can be set in the MobileFirst Server.

Table 11-2. MobileFirst Server properties.

Property	Description	Default Value
`mfp.analytics.console.url`	Set this property to the URL of your MobileFirst Analytics Console. For example, `http://<hostname>:<port>/analytics/console`. Setting this property enables the analytics icon on the MobileFirst Operations Console.	None
`mfp.analytics.logs.forward`	If this property it set to `true`, server logs that are recorded on the MobileFirst Server are captured in MobileFirst Analytics.	`true`
`mfp.analytics.url`	Required. The URL that is exposed by the MobileFirst Analytics Server that receives incoming analytics data. For example, `http://<hostname>:<port>/analytics-service/rest/v2`.	None
`analyticsconsole/mfp.analytics.url`	Optional. Full URI of the Analytics REST services. In a scenario with a firewall or a secured reverse proxy, this URI must be the external URI, not the internal URI inside the local LAN. This value can contain `*` in places of the URI protocol, host name, or port, to denote the corresponding part from the incoming URL.	`*://*:*/analytics-service`, with the protocol, host name, and port dynamically determined
`mfp.analytics.username`	The user name that is used if the data entry point is protected with basic authentication.	None
`mfp.analytics.password`	The password that is used if the data entry point is protected with basic authentication.	None

MobileFirst Analytics Server

The following table shows the properties that can be set in the MobileFirst Analytics Server.

Table 11-3. MobileFirst Analytics Server properties.

Property	Description	Default Value
`analytics/nodetype`	Defines the Elasticsearch node type. Valid values are `master` and `data`. If this property is not set, then the node acts as both a master-eligible node and a data node.	None
`analytics/shards`	The number of shards per index. This value can be set only by the first node that is started in the cluster and cannot be changed.	1
`analytics/replicas_per_shard`	The number of replicas for each shard in the cluster. This value can be changed dynamically in a running cluster.	0
`analytics/masternodes`	A comma-delimited string that contains the host name and ports of the master-eligible nodes.	None
`analytics/clustername`	Name of the cluster. Set this value if you plan to have multiple clusters that operate in the same subset and need to uniquely identify them.	`worklight`
`analytics/nodename`	Name of a node in the cluster.	A randomly generated string
`analytics/datapath`	The path that analytics data is saved to on the file system.	`./analyticsData`
`analytics/settingspath`	The path to an Elasticsearch settings file. For more information, see "Elasticsearch" on page 11-18.	None
`analytics/transportport`	The port that is used for node-to-node communication.	9600
`analytics/httpport`	The port that is used for HTTP communication to Elasticsearch.	9500
`analytics/http.enabled`	Enables or disables HTTP communication to Elasticsearch.	`false`
`analytics/serviceProxyURL`	The analytics UI WAR file and analytics service WAR file can be installed to separate application servers. If you choose to do so, you must understand that the JavaScript run time in the UI WAR file can be blocked by cross-site scripting prevention in the browser. To bypass this block, the UI WAR file includes Java proxy code so that the JavaScript run time retrieves REST API responses from the origin server. But the proxy is configured to forward REST API requests to the analytics service WAR file. Configure this property if you installed your WAR files to separate application servers.	None
`analytics/bootstrap.mlockall`	This property prevents any Elasticsearch memory from being swapped to disk.	`true`
`analytics/multicast`	Enables or disables multicast node discovery.	`false`

Table 11-3. MobileFirst Analytics Server properties (continued).

Property	Description	Default Value
`analytics/warmupFrequencyInSeconds`	The frequency at which warmup queries are run. Warmup queries run in the background to force query results into memory, which improves web console performance. Negative values disable the warmup queries.	`600`
`analytics/tenant`	Name of the main Elasticsearch index.	`worklight`

In all cases where the key does not contain a period (like `httpport` but not `http.enabled`), the setting can be controlled by system environment variables where the variable name is prefixed with `ANALYTICS_`. When both the JNDI property and the system environment variable are set, the system environment variable takes precedence. For example, if you have both the `analytics/httpport` JNDI property and the `ANALTYICS_httpport` system environment variable set, the value for `ANALYTICS_httpport` is used.

Document Time to Live (TTL)

TTL is effectively how you can establish and maintain a data retention policy. Your decisions have dramatic consequences on your system resource needs. The long you keep data, the more RAM, disk, and scaling is likely needed.

Each document type has its own TTL. Setting a document's TTL enables automatic deletion of the document after it is stored for the specified amount of time.

Each TTL JNDI property is named `analytics/TTL_<`*`document type`*`>`. For example, the TTL setting for `NetworkTransaction` is named `analytics/TTL_NetworkTransaction`.

These values can be set by using basic time units as follows.

- `1Y` = 1 year
- `1M` = 1 month
- `1w` = 1 week
- `1d` = 1 day
- `1h` = 1 hour
- `1m` = 1 minute
- `1s` = 1 second
- `1ms` = 1 millisecond

Note: If you are migrating from previous versions of MobileFirst Analytics Server and previously configured any TTL JNDI properties, see “Migration of server properties used by previous versions of MobileFirst Analytics Server” on page 11-12.

Elasticsearch

The underlying storage and clustering technology that serves the MobileFirst Analytics Console is Elasticsearch.

Elasticsearch provides many tunable properties, mostly for performance tuning. Many of the JNDI properties are abstractions of properties that are provided by Elasticsearch.

All properties that are provided by Elasticsearch can also be set by using JNDI properties with `analytics/` prepended before the property name. For example, `threadpool.search.queue_size` is a property that is provided by Elasticsearch. It can be set with the following JNDI property.

```
<jndiEntry jndiName="analytics/threadpool.search.queue_size" value="100" />
```

These properties are normally 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, as follows.

```
<jndiEntry jndiName="analytics/settingspath" value="/home/system/elasticsearch.yml" />
```

Unless you are an expert Elasticsearch IT manager, identified a specific need, or were instructed by your services or support team, do not be tempted to fiddle with these settings.

Backing up Analytics data

Learn about how to back up your MobileFirst Analytics data.

The data for MobileFirst Analytics is stored as a set of files on the MobileFirst Analytics Server file system. The location of this folder is specified by the `datapath` JNDI property in the MobileFirst Analytics Server configuration. For more information about the JNDI properties, see "Configuration properties" on page 11-15.

The MobileFirst 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 is required when you back up these files, other than ensuring that the MobileFirst 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 Analytics Server before you start your backup.

Cluster management and Elasticsearch

Manage clusters and add nodes to relieve memory and capacity strain.

Add a Node to the Cluster

You can add a new node to the cluster by installing the MobileFirst Analytics Server or by running a standalone Elasticsearch instance.

If you choose the standalone Elasticsearch instance, you relieve some cluster strain for memory and capacity requirements, but you do not relieve data ingestion strain. Data reports must always go through the MobileFirst Analytics Server for preservation of data integrity and data optimization prior to going to persistent store.

You can mix and match.

The underlying Elasticsearch data store expects nodes to be homogenous, so do not mix a powerful 8-core 64 GB RAM rack system with a leftover surplus notebook in your cluster. Use similar hardware among the nodes.

Adding a MobileFirst Analytics Server to the cluster:

Learn how to add a MobileFirst Analytics Server to the cluster.

About this task

Because Elasticsearch is embedded in the MobileFirst Analytics Server, and it is responsible for participating in the cluster, do not use the application server's features to define cluster behavior. You do not want to create a WebSphere Application Server Liberty farm, for example. Trust the underlying Elasticsearch run time to participate in the cluster. However, you must configure it properly.

In the following sample instructions, do not configure the node to be a master node or a data node. Instead, configure the node as a "search load balancer" whose purpose is to be up temporarily so that the Elasticsearch REST API is exposed for monitoring and dynamic configuration.

Note: Remember to configure the hardware and operating system of this node according to “System requirements” on page 11-2.

Note: Port 9600 is the transport port that is used by Elasticsearch. Therefore, port 9600 must be open through any firewalls between cluster nodes.

Procedure

1. Install the analytics service WAR file and the analytics UI WAR file (if you want the UI) to the application server on the newly allocated system.

   Install this instance of the MobileFirst Analytics Server to any of the supported app servers.
   - “Installing MobileFirst Analytics on WebSphere Application Server Liberty” on page 11-4
   - “Installing MobileFirst Analytics on Tomcat” on page 11-6
   - “Installing MobileFirst Analytics on WebSphere Application Server” on page 11-7
2. Edit the application server's configuration file for JNDI properties (or use system environment variables) to configure at least the following flags.

Table 11-4. Flags to configure

Flag	Value (example)	Default	Note
`cluster.name`	`worklight`	`worklight`	The cluster that you intend this node to join.
`discovery.zen.ping.multicast.enabled`	`false`	`true`	Set to false to avoid accidental cluster join.

Table 11-4. Flags to configure (continued)

Flag	Value (example)	Default	Note
`discovery.zen.ping.unicast.hosts`	`["9.8.7.6:9600"]`	None	List of master nodes in the existing cluster. Change the default port of 9600 if you specified a transport port setting on the master nodes.
`node.master`	`false`	`true`	Do not allow this node to be a master.
`node.data`	`false`	`true`	Do not allow this node to store data.
`http.enabled`	`true`	`true`	Open unsecured HTTP port 9200 for Elasticsearch REST API.

3. Consider all configuration flags in production scenarios. You might want Elasticsearch to keep the plug-ins in a different file system directory than the data, so you must set the `path.plugins` flag.
4. Run the application server and start the WAR applications if necessary.
5. Confirm that this new node joined the cluster by watching the console output on this new node, or by observing the node count in the **Cluster and Node** section of the **Administration** page in MobileFirst Analytics Console.

Adding a stand-alone Elasticsearch node to the cluster:

Learn how to add a stand-alone Elasticsearch node to the cluster.

About this task

You can add a stand-alone Elasticsearch node to your existing MobileFirst Analytics cluster in just a few simple steps. However, you must decide the role of this node. Is it going to be a master-eligible node? If so, remember to avoid the split-brain issue from "Configuration guide" on page 11-14. Is it going to be a data node? Is it going to be a client-only node? Perhaps you want a client-only node so that you can start a node temporarily to expose Elasticsearch's REST API directly to affect dynamic configuration changes to your running cluster.

In the following sample instructions, do not configure the node to be a master node or a data node. Instead, configure the node as a "search load balancer" whose purpose is to be up temporarily so that the Elasticsearch REST API is exposed for monitoring and dynamic configuration.

Note: Remember to configure the hardware and operating system of this node according to "System requirements" on page 11-2.

Note: Port 9600 is the transport port that is used by Elasticsearch. Therefore, port 9600 must be open through any firewalls between cluster nodes.

Procedure

1. Download Elasticsearch from https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-1.7.5.tar.gz.

2. Decompress the file.
3. Edit the `config/elasticsearch.yml` file and configure at least the following flags.

Table 11-5. Flags to configure

Flag	Value (example)	Default	Note
`cluster.name`	`worklight`	`worklight`	The cluster that you intend this node to join.
`discovery.zen.ping.multicast.enabled`	`false`	`true`	Set to false to avoid accidental cluster join.
`discovery.zen.ping.unicast.hosts`	`["9.8.7.6:9600"]`	None	List of master nodes in the existing cluster. Change the default port of 9600 if you specified a transport port setting on the master nodes.
`node.master`	`false`	`true`	Do not allow this node to be a master.
`node.data`	`false`	`true`	Do not allow this node to store data.
`http.enabled`	`true`	`true`	Open unsecured HTTP port 9200 for Elasticsearch REST API.

4. Consider all configuration flags in production scenarios. You might want Elasticsearch to keep the plug-ins in a different file system directory than the data, so you must set the `path.plugins` flag.
5. Run `./bin/plugin -i elasticsearch/elasticsearch-analytics-icu/2.7.0` to install the ICU plug-in.
6. Run `./bin/elasticsearch`.
7. Confirm that this new node joined the cluster by watching the console output on this new node, or by observing the node count in the **Cluster and Node** section of the **Administration** page in MobileFirst Analytics Console.

Circuit breakers:

Learn about Elasticsearch circuit breakers.

Elasticsearch contains multiple circuit breakers that are used to prevent operations from causing an `OutOfMemoryError`. For example, if a query that serves data to the MobileFirst Operations Console results in using 40% of the JVM heap, the circuit breaker triggers, an exception is raised, and the console receives empty data.

Elasticsearch also has protections for filling up the disk. If the disk on which the Elasticsearch data store is configured to write fills to 90% capacity, the Elasticsearch node informs the master node in the cluster. The master node then redirects new document-writes away from the nearly full node. If you have only one node in your cluster, no secondary node to which data can be written is available. Therefore, no data is written and is lost.

Configuring analytics from the MobileFirst Operations Console

Configure server-side analytics runtime behavior, define reporting, and view reports from the MobileFirst Operations Console and the MobileFirst Analytics Console.

The MobileFirst Operations Console and MobileFirst Analytics Console enable server-side analytics configuration and provides report setup and viewing.

Enabling or disabling data collection from the MobileFirst Operations Console

After MobileFirst Analytics is installed and configured for your application server, you can enable or disable data collection from the MobileFirst Operations Console.

Before you begin

1. Install and configure MobileFirst Analytics for your application server. For more information, see "MobileFirst Analytics Server installation guide" on page 11-2.
2. Open the MobileFirst Operations Console as explained in "Opening the MobileFirst Operations Console" on page 7-12.

About this task

When you first open the console, the Dashboard view shows the current state of the selected runtime. In the Runtime status table, Analytics is displayed as active. By default, the collection of data for analysis by the Analytics server is enabled. You can disable it, for example to save processing time.

Note: Only the **`mfpadmin`** administrator role and the **`mfpdeployer`** deployer role are allowed to disable or re-enable data collection for Analytics. For more information about users and roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

Procedure

1. In the navigation sidebar, click **Runtime settings**.

   To avoid inadvertent changes, runtime properties displayed in read-only mode.
2. To make the settings editable, click the **Edit** button.

   If you logged in with a role other than administrator or deployer, the **Edit** button is not visible because you are not allowed to modify runtime properties.
3. From the **Data collection enabled** drop-down menu, select **false** to disable data collection.
4. Click **Save**.
5. Click the **Read Only** button to lock the properties again.

Role-based access control

Content in the MobileFirst Analytics Console is restricted by predefined security roles.

The MobileFirst Analytics Console shows different content based on the security role of the logged-in user. The following table shows the security role and the access that is granted to it in the MobileFirst Analytics Console.

Table 11-6. Role-based access for the MobileFirst Analytics Console.

Role	Role name	Viewing Access	Editing Access
Administrator	`analytics_administrator`	Everything.	Everything.
Infrastructure	`analytics_infrastructure`	Everything.	Custom Charts and Alerts pages.
Developer	`analytics_developer`	Everything except for the Administration pages.	Custom Charts and Alerts pages.
Support	`analytics_support`	Everything except for the Administration pages.	Custom Charts and Alerts pages.
Business	`analytics_business`	Everything except for the Administration and Infrastructure pages.	Custom Charts and Alerts pages.

For information on setting up roles, see "Configuring user authentication for MobileFirst Server administration" on page 6-166.

Setting Log Filters from the MobileFirst Operations Console

From the MobileFirst Operations Console, you create, edit, and delete client log filters on a registered application. Log filters take effect if the application includes specific method calls.

About this task

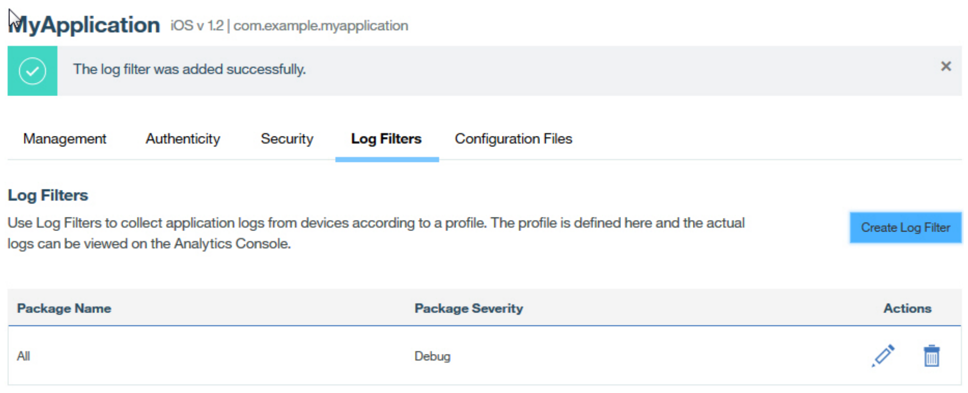
For any application that is registered on MobileFirst Operations Console, you can configure the verbosity of the log level that the client device can capture and send back to MobileFirst Server using log filters.

Procedure

1. Under **Runtimes** > **Applications**, select the application for which you want to configure the log level.
2. Click the **Log Filters** tab.
3. In the Create Log Filter dialog, follow the contextual instructions and click **Save** when you are finished.

   You can edit an existing log filter by clicking the pencil-shaped icon to open the Edit Log Filter dialog.

   A success message is displayed.



MyApplication iOS v 1.2 | com.example.myapplication

The log filter was added successfully.

Log Filters

Use Log Filters to collect application logs from devices according to a profile. The profile is defined here and the actual logs can be viewed on the Analytics Console.

Package Name	Package Severity	Actions
All	Debug	

Results

Adding, modifying, or deleting a log filter from the MobileFirst Operations Console does not take effect on the device in real time. These changes take effect only if the developer has coded the client app to pull the client log in the application, by using either of the following API calls, provided by the SDK (see “Fetching server configuration profiles” on page 11-42).

Each time the filters are retrieved, the device is synchronized with the log profile. A good practice is to put the appropriate API call in the application code path that gets executed often, for example when the application starts or when application foreground events occur.

What to do next

You can later retrieve or view client logs on the MobileFirst Analytics Console.

Custom charts

Learn about custom charts.

Custom chart creation

The MobileFirst Analytics Console offers the ability to create custom charts. You can extend and supplement the existing default charts that are provided in the MobileFirst Analytics Console.

Creating a custom chart

You can create custom charts for the following event types.

- App Session
- Network Transactions
- Push Notifications
- Client Logs
- Server Logs
- Custom Data

The custom charts creation builder takes you through four main stages. To begin the custom charts creation process, click **Create Chart** on the **Custom Charts** page of the **Dashboard** section in the MobileFirst Analytics Console.

General Settings

The **General Settings** tab contains three fields.

- **Chart Title** - the title for the chart (defaults to **Untitled Chart**).
- **Event Type** - the event type to be visualized.
- **Chart Type** - the chart type. For more information, see “Chart types” on page 11-26.

After you select the **Event Type** and **Chart Type**, the **Chart Definition** tab appears.

Chart Definition

Use the **Chart Definition** tab to define the chart for the specified chart type that you selected in the **General Settings** tab. After you define the chart, you can set the chart filters and chart properties.

Chart Filters

Use **Chart Filters** to fine-tune your custom chart. For example, if you are interested in seeing the average app session duration for a particular app, you can specify the following options.

- On the **Chart Filters** tab, select **Application Name** for **Property**.
- Select **Equals** for **Operator**.
- Select the name of your app for **Value**.
- Click **Add Filter**.

The app name filter is added to the table of filters for your chart. Multiple filters can be defined for any chart.

Chart Properties

Chart properties are available for the **Table**, **Bar Graph**, and **Line Graph** chart types. The goal of chart properties is to enhance how the data is presented so that the visualization is more effective.

If you created a **Table** chart, the chart properties can be set to define the table page size, the field on which to sort, and the sort order of the field.

If you created a **Bar Graph** or **Line Graph** chart, the chart properties can be set to label threshold lines to add a frame of reference for anyone who is monitoring the chart.

Chart types

You can create a different types of custom chart to visualize data.

Bar Graph

The bar graph allows for visualization of numeric data over an x-axis. When you define a bar graph, you must choose the value for **X-Axis** first. You can choose from the following possible values.

- **Timeline** - choose **Timeline** for **X-Axis** if you want to see your data as a trend (for example, average app session duration over time).
- **Property** - choose **Property** if you want to see a count breakdown for the specific property. If you choose **Property** for **X-Axis**, then **Total** is implicitly chosen for **Y-Axis**. For example, choose **Property** for **X-Axis** and **Application Name** for **Property** to see a count for a specified event type, which is broken down by app name.

After you define a value for **X-Axis**, you can define a value for **Y-Axis**. If you choose **Timeline** for **X-Axis**, you can choose the following possible values for **Y-Axis**.

- **Average** - averages a numeric property in the supplied event type.
- **Total** - a total count of a property in the supplied event type.
- **Unique** - a unique count of a property in the supplied event type.

After you define the chart axes, you must choose a value for **Property**.

Line Graph

The line graph allows for the visualization of some metric over time. This type of chart is valuable when you want to visualize data in terms of a trend over time. The first value to define when you create a line graph is **Measure**, which has the following possible values.

- **Average** - averages a numeric property in the supplied event type.
- **Total** - a total count of a property in the supplied event type.
- **Unique** - a unique count of a property in the supplied event type.

After you define the measurement, you must choose a value for **Property**.

Flow Chart

The flow chart allows for the visualization of flow breakdown of one property to another. For a flow chart, the following properties must be set.

- **Source** - the value of a source node in the diagram.
- **Destination** - the value of the destination node in the diagram.
- **Property** - a property value from either the source node or the destination node.

With the flow chart, you can see the density breakdown of various sources that flow to a destination, or vice versa. For example, if you want to see the breakdown of log severities for an app, you can define the following values.

- Select **Application Name** for **Source**.
- Select **Log Level** for **Destination**.
- Select the name of your app for **Property**.

Metric Group

The metric group can be used to visualize a single metric that is measured as either an average value, a total count, or a unique count. To define a metric group, you must define one of the following possible values for **Measure**.

- **Average** - averages a numeric property in the supplied event type.
- **Total** - a total count of a property in the supplied event type.
- **Unique** - a unique count of a property in the supplied event type.

After you define the measurement, you must choose a value for **Property**. This metric is displayed in the metric group.

Pie Chart

The pie chart can be used to visualize the count breakdown of values for a particular property. For example, if you want to see a crash breakdown, define the following values.

- Select **App Session** for **Event Type**.
- Select **Pie Chart** for **Chart Type**.
- Select **Closed By** for **Property**.

The resulting pie chart shows the breakdown of app sessions that were closed by the user as opposed to app sessions that were closed by a crash.

Table

The table is useful when you want to see the raw data. Building a table is as simple as adding columns for the raw data that you want to see.

Because not all properties are required for specific event types, null values can appear in your table. If you want to prevent these rows from appearing in your table, add an **Exists** filter for a specific property in the **Chart Filters** tab.

Creating custom charts for client logs

You can create a custom chart for client logs that contain log information that is sent with the platform's `Logger` API. The log information also includes contextual information about the device, including environment, app name, and app version.

Before you begin

You must log custom events to populate custom charts. For information on sending custom events from the client app, see "Capturing custom data" on page 11-36.

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 app. 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

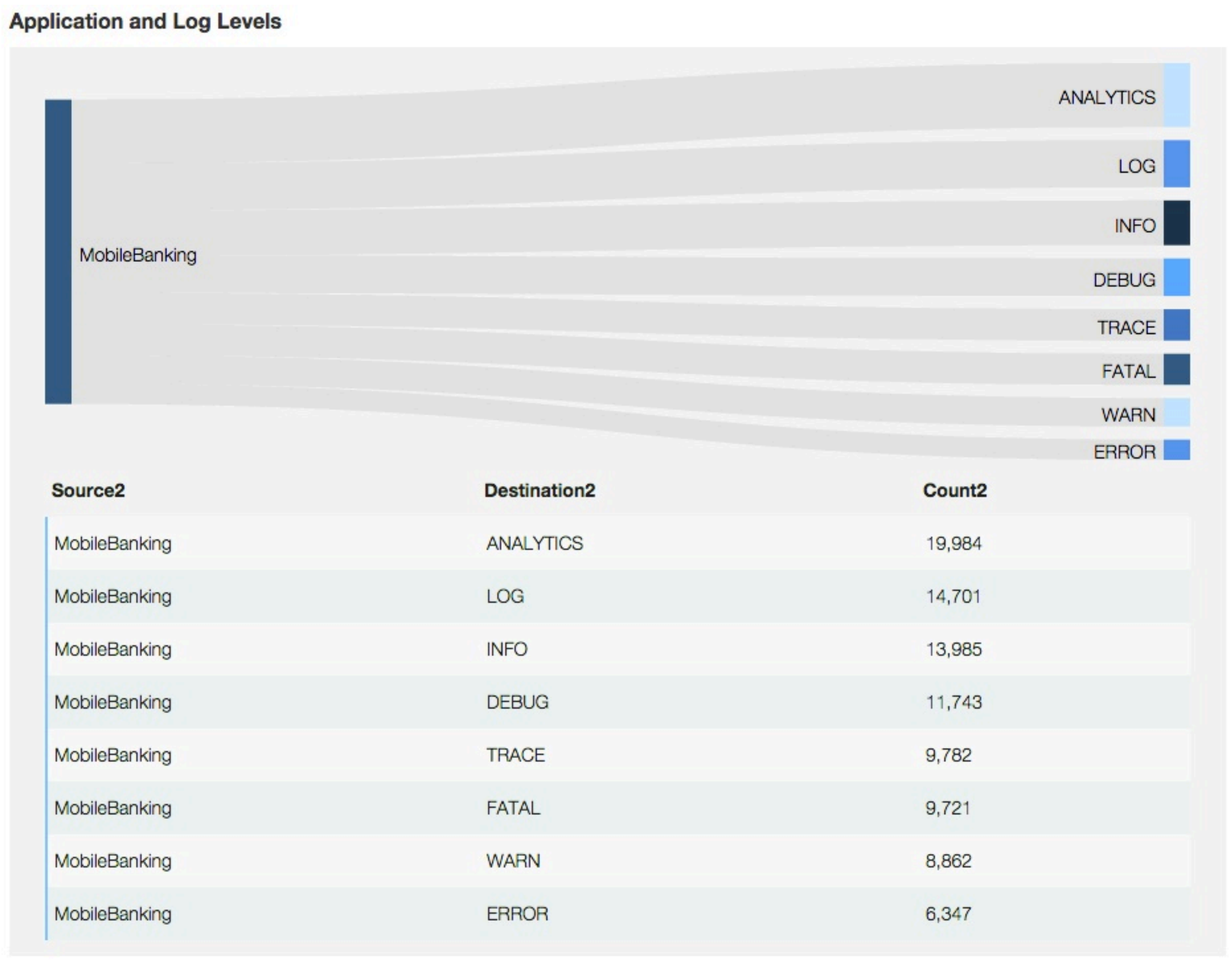
Procedure

1. From the client app, populate the data by sending captured logs to the server. See "Sending captured logs" on page 11-39.
2. 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.
3. Click **Create Chart** to create a new custom chart.
4. Provide the following values:
   - **Chart Title:** Application and Log Levels
   - **Event Type:** Client Logs
   - **Chart Type:** Flow Chart
5. Click the **Chart Definition** tab.
6. Provide the following values:

- **Source:** Application Name
- **Destination:** Log Level
- **Property:** *your app name*

7. Click **Save**.

Results



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 11-1. Application and Log Levels

Exporting custom chart data

You can download the data that is shown for any custom chart.

About this task

To download custom chart data, choose one of the following steps. At the beginning of each custom chart, the following icons are displayed.

- **Export with URL** - looks like a chain link
- **Download Chart** - looks like a down arrow
- **Edit Chart** - looks like a pencil
- **Delete Chart** - looks like a trashcan

Procedure

1. Optional: Click the **Dowload Chart** icon to download a file in JSON format from the MobileFirst Analytics Console.
2. Optional: Click the **Export with URL** icon to generate an export link from the MobileFirst Analytics Console to call from an HTTP client. This option is useful if you want to write a script to automate the export processes on a specified time interval.

Exporting and importing custom chart definitions

You can export and import custom chart definitions in the MobileFirst Analytics Console. If you are moving from a test environment to a production deployment, you can save time by exporting your custom chart definitions instead of re-creating your custom charts on your new cluster.

Before you begin

Ensure that you have at least one custom chart in the MobileFirst Analytics Console.

About this task

To duplicate your custom charts, follow these steps.

Procedure

1. Click the **Custom Charts** tab in the **Dashboard** section in the MobileFirst Analytics Console.
2. Click **Export Charts** to download a JSON file with your chart definition.
3. Choose a location to save the JSON file.
4. Click **Import Charts** to import your JSON file. 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.

Alerts

You can set reactive thresholds in the MobileFirst Analytics Console to trigger alerts when a specific criteria is met. This feature provides a proactive means to truly monitor the health of your mobile apps without having to check the MobileFirst Analytics Console regularly.

You can set thresholds at a broad level (a specific app) or at a granular level (a specific app instance or device). Alert notifications can be configured to display in the MobileFirst Analytics Console, and also be sent to a pre-configured REST endpoint.

Creating an alert definition for app crashes

You can create an alert definition based on app crashes.

Before you begin

Ensure that the MobileFirst Analytics Server is started and ready to receive client logs.

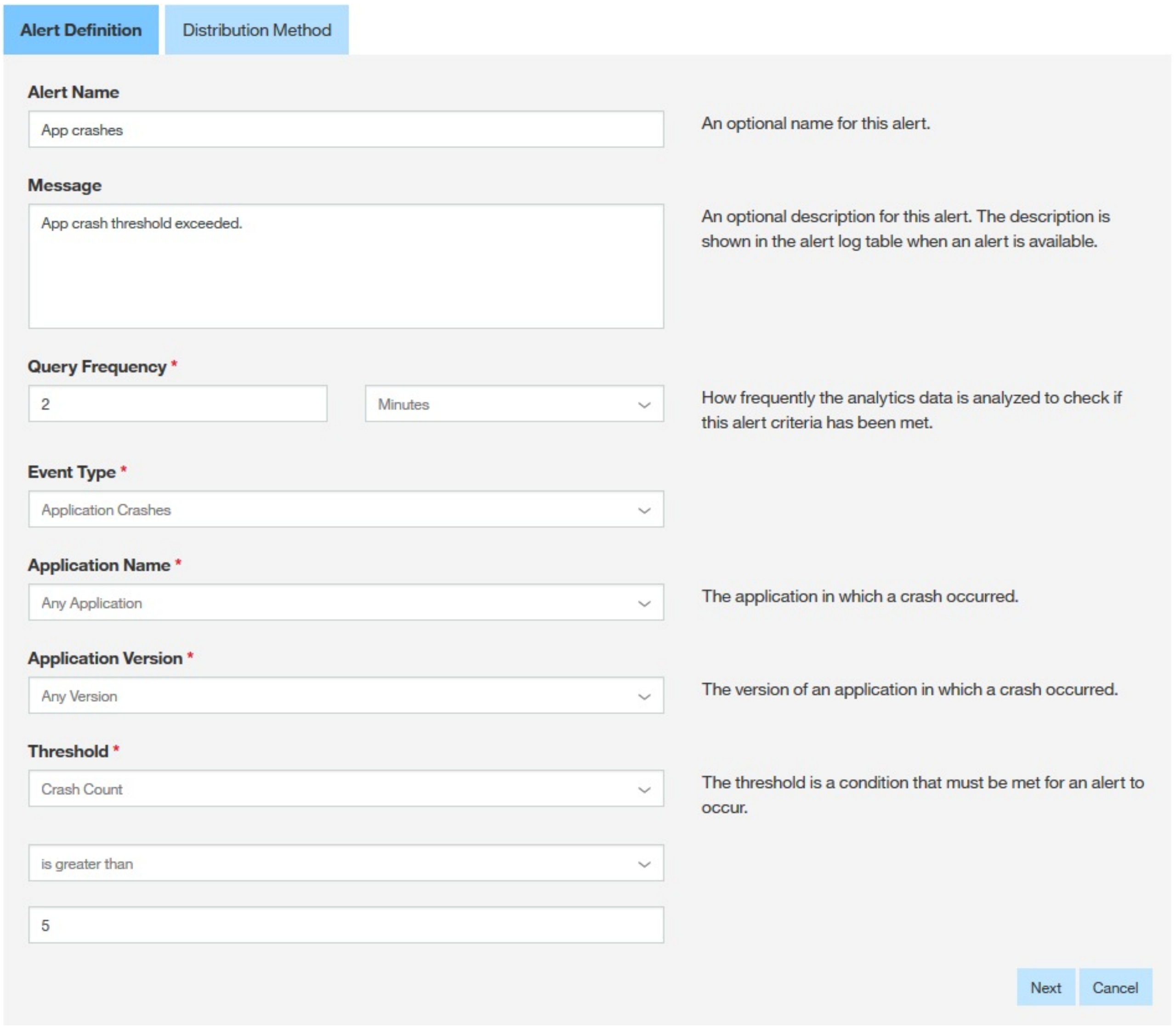
About this task

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 app that crashed 5 or more times.

Procedure

1. In the MobileFirst Analytics Console, click the **Alert Management** tab.
2. 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

App crashes

An optional name for this alert.

Message

App crash threshold exceeded.

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

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.

Custom webhook

You can set up a distribution method for your alert. One option is to define a custom web hook to which a payload is sent, when an alert threshold is triggered. You can also specify a set of optional headers, and basic auth credentials if your endpoint is protected by basic auth.

By default, the POST request has a Content-Type of `application/json`. The following example shows a sample payload.

```
{
  "timestamp": 1442848504431,
  "condition": {"value":5.0,"operator":"GTE"},
  "value": "CRASH",
  "offenders": [
    { "XXX 1.0": 5.0 },
    { "XXX 2.0": 1.0 }
  ],
  "property":"closedBy",
  "eventType":"MfpAppSession",
  "title":" Crash Count Alert for Application ABC",
  "message": "The crash count for a application ABC exceeded XYZ.
    View the Crash Summary table in the Crashes tab in the Apps
    section of the MobileFirst Analytics Console
    to see a detailed stacktrace of this crash instance."
}
```

The POST request includes the following attributes.

- **timestamp** - the time at which the alert notification was created.
- **condition** - the threshold that was set by the user (for example, greater than or equals 5).
- **eventType** - the `eventType` that was queried.
- **property** - the property of the `eventType` that was queried.
- **value** - the value of the property that was queried.
- **offenders** - a list of apps or devices that triggered the alert.
- **title** - the user-defined title.
- **message** - the user-defined message.

Viewing alert details

You can view the details of your triggered alerts.

Before you begin

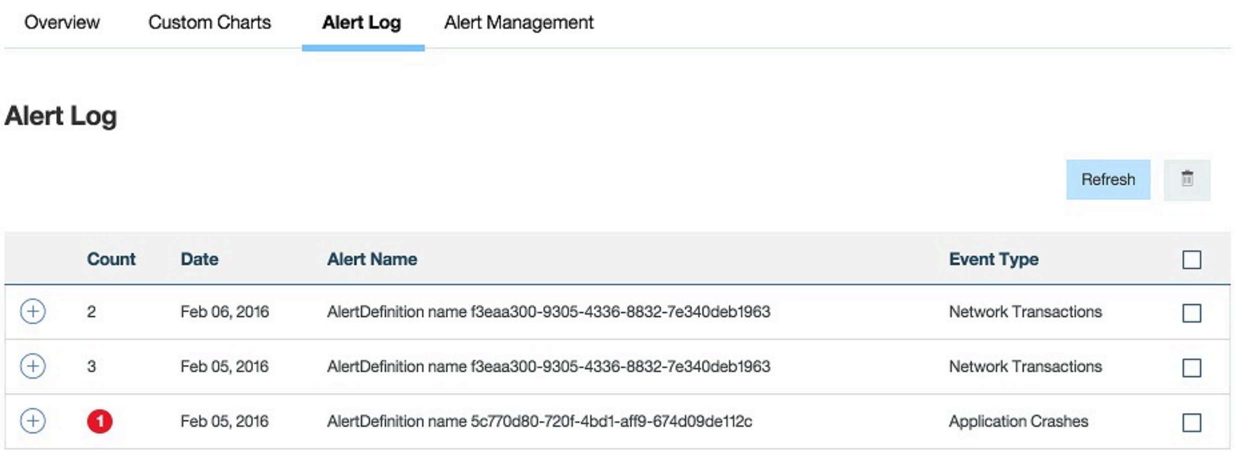
Ensure that the MobileFirst Analytics Server is started and ready to receive analytics data.

About this task

In this example, you view the details of your triggered alerts from the **Alerts Log** tab.

Procedure

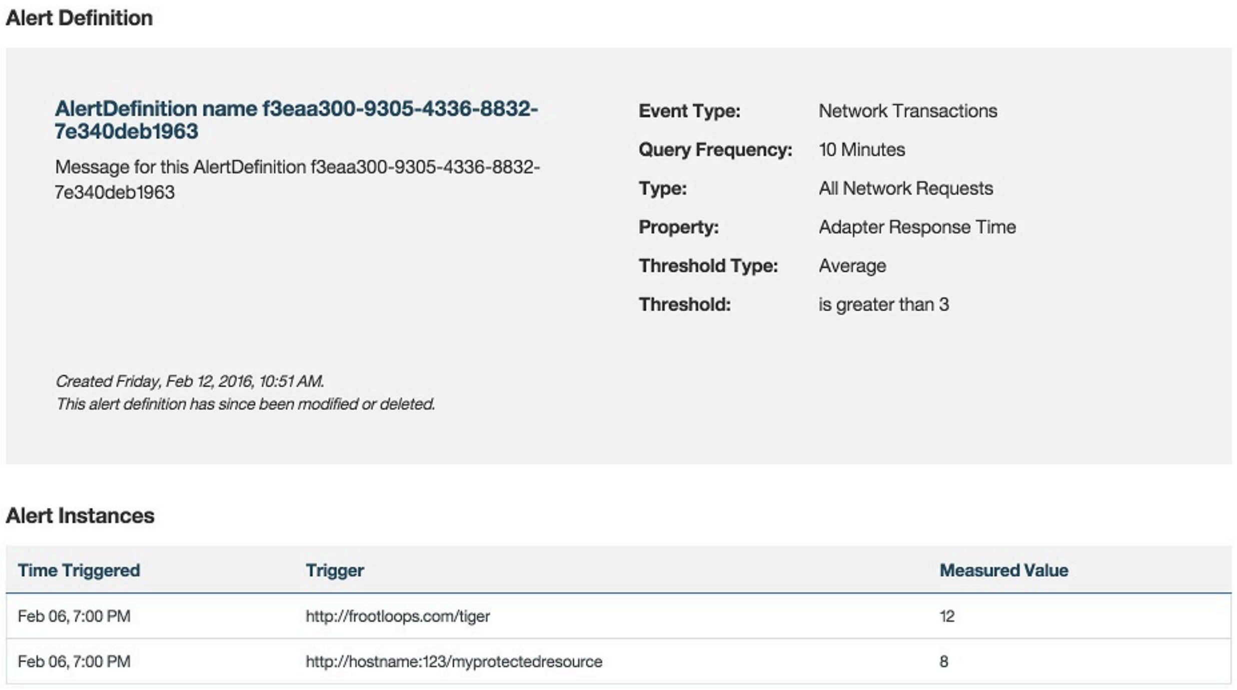
1. In the MobileFirst Analytics Console, open the **Alerts Log** tab :



Alert Log

	Count	Date	Alert Name	Event Type	☐
⊕	2	Feb 06, 2016	AlertDefinition name f3eaa300-9305-4336-8832-7e340deb1963	Network Transactions	☐
⊕	3	Feb 05, 2016	AlertDefinition name f3eaa300-9305-4336-8832-7e340deb1963	Network Transactions	☐
⊕	1	Feb 05, 2016	AlertDefinition name 5c770d80-720f-4bd1-aff9-674d09de112c	Application Crashes	☐

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 f3eaa300-9305-4336-8832-7e340deb1963

Message for this AlertDefinition f3eaa300-9305-4336-8832-7e340deb1963

Event Type:	Network Transactions
Query Frequency:	10 Minutes
Type:	All Network Requests
Property:	Adapter Response Time
Threshold Type:	Average
Threshold:	is greater than 3

Created Friday, Feb 12, 2016, 10:51 AM.
This alert definition has since been modified or deleted.

Alert Instances

Time Triggered	Trigger	Measured Value
Feb 06, 7:00 PM	http://frootloops.com/tiger	12
Feb 06, 7:00 PM	http://hostname:123/myprotectedresource	8

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.

Developing the analytics client

Capture analytics data and send it to the MobileFirst Analytics Server by using analytics client SDKs.

Analytics SDK

Use the Analytics SDK to capture and send analytics data.

Capturing analytics

You can initialize the MobileFirst Analytics SDK and enable the capture of lifecycle and network analytic data.

About this task

The MobileFirst Analytics API allows for the capturing of the following metrics.

- App lifecycle events - app usage rates, usage duration, app crash rates
- Network usage - breakdown of API call frequencies, network performance metrics
- Users - users of the app that are identified by a supplied user ID
- Custom analytics - custom key/value pairs that are defined by the app developer

The initialization of the analytics API must be written in native code, even in Cordova apps.

- To capture app usage, you must register app lifecycle event listeners before the event to which you are listening occurs.
- To use the file system or native language and device features, the API must be initialized. If the API is used in a way that requires native device features (like the file system), but was not initialized, the API call fails. This behavior is especially true on Android.

Note: No listeners are required for the web apps.

To initialize the MobileFirst Analytics SDK and enable the capture of lifecycle and network analytic data:

- On iOS, add the following code in your Application Delegate `application:didFinishLaunchingWithOptions` method.

  ```
  WLAnalytics *analytics = [WLAnalytics sharedInstance];
  [analytics addDeviceEventListener:NETWORK];
  [analytics addDeviceEventListener:LIFECYCLE];
  ```
- Similarly, on Android, add the following code in your Application subclass `onCreate` method.

  ```
  WLAnalytics.init(this);
  WLAnalytics.addDeviceEventListener(DeviceEvent.NETWORK);
  WLAnalytics.addDeviceEventListener(DeviceEvent.LIFECYCLE);
  ```
- For Cordova apps, the listener must be created in the native code. See “MobileFirst Cordova plug-in initialization for analytics” on page 7-124 and “MobileFirst Cordova plug-in initialization of analytics” on page 7-125.
- For web apps, no listeners are required. Analytics can be enabled and disabled through the `ibmmfpfanalytics.logger` class.

  ```
  ibmmfpfanalytics.logger.config({analyticsCapture: true});
  ```

Tracking users:

To track individual users, use the `setUserContext` and `unsetUserContext` methods.

About this task

To track users, follow these steps.

- **iOS**
  - Add the following code when the user logs in.
    ```
    [[WLAnalytics sharedInstance] setUserContext:@"John Doe"];
    ```
  - Add the following code when the user logs out.
    ```
    [[WLAnalytics sharedInstance] unsetUserContext];
    ```
- **Android**
  - Add the following code when the user logs in.
    ```
    WLAnalytics.setUserContext("John Doe");
    ```
  - Add the following code when the user logs out.
    ```
    WLAnalytics.unsetUserContext();
    ```
- **Cordova**

  Not supported.
- **Web**
  - Add the following code when the user logs in.
    ```
    ibmmfpfanalytics.setUserContext(user);
    ```
    There is no `unsetUserContext` in the web API. The user session ends after 30 minutes of inactivity, unless another call is made to ibmmfpfanalytics.setUserContext(user).

Capturing custom data:

You can instrument your app code to capture custom analytics.

About this task

An example demonstrates how to track page transitions within your app.

- On iOS, add the following code in the `ViewController`'s `viewDidLoad` method.
  ```
  NSArray *viewControllers = self.navigationController.viewControllers;
  NSUInteger numViewControllers = viewControllers.count;
  if (numViewControllers >= 2) {
    UIViewController *previous = [viewControllers objectAtIndex:(numViewControllers - 2)];
    NSDictionary *metadata = @{@"fromPage": previous.title, @"toPage": self.title};
    [[WLAnalytics sharedInstance] log:@"page transition" withMetadata:metadata];
  }
  ```
- On Android, add the following code in an `Activity` subclass `onResume()` method.
  ```
  String fromActivity = getIntent().getStringExtra("fromActivity");
  JSONObject metadata = new JSONObject();
  metadata.put("fromPage", fromActivity);
  metadata.put("toPage", "This Page");
  WLAnalytics.log("page transition", metadata);
  ```
- In JavaScript (Cordova), add the following code.
  ```
  // before showing the element identified by 'setting'
  $(document).on('pagebeforeshow', '#setting', function(e, obj) {
    WL.Analytics.log('page transition', {
  ```

```
      'fromPage': obj.prevPage[0].id,
      'toPage': obj.toPage[0].id
    });
  });
```

- For the web API, custom data is sent with the addEvent method.

  ```
  ibmmfpfanalytics.addEvent({'Purchases':'radio'});
  ibmmfpfanalytics.addEvent({'src':'App landing page','target':'About page'});
  ```

Sending analytics

To see client-side analytic data in the MobileFirst Analytics Console, send analytics to the MobileFirst Analytics Server by calling the send method.

Before you begin

Ensure that you enabled the capturing of relevant events. For more information, see "Capturing analytics" on page 11-35.

About this task

Once events are captured by the client, send them periodically to the server. Sending captured analytics to the server can be done explicitly or periodically. The following example shows how to send analytics at 1-minute intervals. Sending data at regular intervals ensures that you are seeing up-to-date analytic data in the MobileFirst Analytics Console.

- **iOS**

  ```
  outputclass="prettyprint">[NSTimer scheduledTimerWithTimeInterval:60
    target:[WLAnalytics sharedInstance]
    selector:@selector(send)
    userInfo:nil
    repeats:YES];
  ```

- **Android**

  ```
  Timer timer = new Timer();
  timer.schedule(new TimerTask() {
    @Override
    public void run() {
      WLAnalytics.send();
    }
  }, 0, 60000);
  ```

- **JavaScript (Cordova)**

  ```
  setInterval(function() {
    WL.Analytics.send();
  }, 60000);
  ```

- **web**

  ```
  setInterval(function() {
    ibmmfpfanalytics.send();
  }, 60000);
  ```

Logger SDK

Learn about the Logger SDK.

It is often difficult to reproduce problems that might occur in the field. Replicating the exact environment and device can be troublesome in these situations. It is helpful to be able to retrieve debug logs from client devices as the problems occur in the environment in which they happen. The MobileFirst client-side Logger API allows developers to instrument their code at varying verbosities. These logs then can be captured, and sent to the MobileFirst Analytics Console for further inspection.

From the client you can send logger requests to the MobileFirst Analytics Server at any logging level. However, the server configuration controls what level of logging requests are allowed. Requests sent below this threshold are ignored.

Logging levels need to be controlled to balance two needs: the need to collect information and the need to limit the quantity of data to fit limited storage ability.

Note: For the web API the browser's local storage significantly limits the amount of logs that can be stored before sending to the server. Therefore the developer must either limit the amount of data sent to the logger, or send smaller data to the server more frequently.

Enabling log capture

By default, log capture is enabled. Log capture saves logs to the client and can be enabled or disabled programmatically. Logs are sent to the server with an explicit send call, or with auto log

About this task

Logging can be enabled or disabled from the client.

Note: Enabling log capture at verbose levels can impact the consumption of the device CPU, file system space, and the size of the payload when the client sends logs over the network.

- **iOS**

  To enable:
  ```
  [OCLogger setCapture:YES];
  ```
  To disable:
  ```
  [OCLogger setCapture:NO];
  ```
- **Android**

  To enable:
  ```
  Logger.setCapture(true);
  ```
  To disable:
  ```
  Logger.setCapture(false);
  ```
- **Cordova JavaScript**

  To enable:
  ```
  WL.Logger.config({ capture: true });
  ```
  To disable:
  ```
  WL.Logger.config({ capture: false });
  ```
- **web**

  To enable:
  ```
  ibmmfpfanalytics.logger.enable(true);
  ```
  To disable:
  ```
  ibmmfpfanalytics.logger.enable(false);
  ```

Adjusting log verbosity

Seven log levels are available within the Logger API.

About this task

The logging levels from the most verbose to the least are as follows:

- TRACE - used for method entry and exit points

- DEBUG - used for method result output
- LOG - used for class instantiation
- INFO - used for reporting initialization
- WARN - used to log deprecated usage warnings
- ERROR - used for unexpected exceptions
- FATAL - used for unrecoverable crashes or hangs

The client SDKs are configured at the FATAL verbosity by default, which means little or no raw debug logs are output or captured. Adjust the verbosity programmatically. Log verbosity can also be adjusted by setting a configuration profile on the MobileFirst Operations Console, which must be retrieved explicitly by your app. Log levels set programatically are valid for the entire app. Levels set by the server are set per package. For more information, see “Fetching server configuration profiles” on page 11-42.

Once logging level is set, either by setting the client or retrieving the server profile, the client filters the logging messages it sends. If a message below the threshold is explicitly sent, the client ignores it.

To set the verbosity level to DEBUG:

- **iOS**
  ```
  [OCLogger setLevel:OCLogger_DEBUG];
  ```
- **Android**
  ```
  Logger.setLevel(Logger.LEVEL.DEBUG);
  ```
- **JavaScript (Cordova)**
  ```
  WL.Logger.config({ level: 'DEBUG' });
  ```
- **web**

  For the web SDK the default `trace` level cannot be changed from the client.

Sending captured logs

Send logs to the server according to your application's logic. **Auto log send** can also be enabled to automatically send logs. If logs are not sent before the maximum size is reached, the log file is then purged in favor of newer logs.

Before you begin

- Ensure that you enabled log capture. For more information, see “Enabling log capture” on page 11-38.
- Ensure that the verbosity of logs that you want to see in the MobileFirst Analytics Console matches the verbosity that is set on the client. See “Setting Log Filters from the MobileFirst Operations Console” on page 11-24.

About this task

As an example to send logs on a 1-minute interval timer, follow these steps.

Note: Adopt the following pattern when you collect log data. Sending data on an interval ensures that you are seeing your log data in near real-time in the MobileFirst Analytics Console.

- **iOS**

```
[NSTimer scheduledTimerWithTimeInterval:60
  target:[OCLogger class]
  selector:@selector(send)
  userInfo:nil
  repeats:YES];
```

- **Android**

  ```
  Timer timer = new Timer();
  timer.schedule(new TimerTask() {
    @Override
    public void run() {
      Logger.send();
    }
  }, 0, 60000);
  ```
- **JavaScript (Cordova)**

  ```
  setInterval(function() {
    WL.Logger.send();
  }, 60000);
  ```
- **Web**

  ```
  setInterval(function() {
     ibmmfpfanalytics.logger.send();
  }, 60000);
  ```

To ensure that all captured logs are sent, consider one of the following strategies:
- Call the `send` method at a time interval.
- Call the `send` method from within the app lifecycle event callbacks.
- Increase the max file size of the persistent log buffer (in bytes):
  - **iOS**

    `[OCLogger setMaxFileSize:150000];`
  - **Android**

    `Logger.setMaxFileSize(150000);`
  - **JavaScript**

    `WL.Logger.config({ maxFileSize: 150000 });`
  - **web**

    The maximum file size for the web API is 5 mb and cannot be changed.

Auto log send

By default, auto log send is enabled. Each time a successful resource request is sent to the server, the captured logs are also sent, with a 60-second minimum interval between sends.

About this task

Auto log send can be enabled or disabled from the client. By default auto log send is enabled.
- **iOS**

  To enable:

  `[OCLogger setAutoSendLogs:YES];`

  To disable:

  `[OCLogger setAutoSendLogs:NO];`
- **Android**

  To enable:

  `Logger.setAutoSendLogs(true);`

  To disable:

```
Logger.setAutoSendLogs(false);
```

- **Cordova JavaScript**

  To enable:

  ```
  WL.Logger.config({autoSendLogs: true});
  ```

  To disable:

  ```
  WL.Logger.config({autoSendLogs: false});
  ```

- **web**

  To enable:

  ```
  ibmmfpfanalytics.enableAutoSend(true);
  ```

  To disable:

  ```
  ibmmfpfanalytics.enableAutoSend(false);
  ```

Fine-tuning with the Logger API

The MobileFirst client-side SDK makes internal use of the Logger API. By default, you are capturing log entries made by the SDK. To fine-tune log collection, use logger instances with package names. You can also control which logging level is captured by the analytics using server-side filters.

About this task

As an example to capture logs only where the level is ERROR for the `myApp` package name, follow these steps.

- **iOS**
  1. Use a logger instance with the `myApp` package name.

     ```
     OCLogger *logger = [OCLogger getInstanceWithPackage:@"MyApp"];
     ```

  2. Specify a filter to restrict log capture and log output to only the specified level and package programmatically.

     ```
     [OCLogger setFilters:@{@"MyApp": @(OCLogger_ERROR)}];
     ```

  3. **Optional:** Control the filters remotely by following the steps in “Fetching server configuration profiles” on page 11-42.
- **Android**
  1. Use a logger instance with the `myApp` package name.

     ```
     Logger logger = Logger.getInstance("MyApp");
     ```

  2. Specify a filter to restrict log capture and log output to only the specified level and package programmatically.

     ```
     HashMap<String, LEVEL> filters = new HashMap<>();
     filters.put("MyApp", LEVEL.ERROR);
     Logger.setFilters(filters);
     ```

  3. **Optional:** Control the filters remotely by following the steps in “Fetching server configuration profiles” on page 11-42.
- **JavaScript (Cordova)**
  1. Use a logger instance with the `myApp` package name.

     ```
     var logger = WL.Logger.create({ pkg: 'MyApp' });
     ```

  2. **Optional:** Specify a filter to restrict log capture and log output to only the specified level and package programmatically.

     ```
     WL.Logger.config({
       filters: {
         'MyApp': 'ERROR'
       }
     });
     ```

3. **Optional:** Control the filters remotely by following the steps in "Fetching server configuration profiles."

- **web**

  For the web SDK the level cannot be set by the client. All logging is sent to the server until the client retrieves the server profile.

Fetching server configuration profiles

Logging level can be set by retrieving server configuration files.

About this task

Logging levels can be set by the client (see "Adjusting log verbosity" on page 11-38) or by retrieving configuration profiles from the server. From the MobileFirst Operations Console, a log level can be set globally (all logger instances) or for a specific package or packages. For the client to fetch the configuration overrides that are set on the server, the `updateConfigFromServer` method must be called from a place in the code that is regularly run, such as in the app lifecycle callbacks.

To call the `updateConfigFromServer` method, follow these steps.

- **iOS**
  ```
  [OCLogger updateConfigFromServer];
  ```
- **Android**
  ```
  Logger.updateConfigFromServer();
  ```
- **Cordova (JavaScript)**
  ```
  WL.Logger.updateConfigFromServer();
  ```
- **web (JavaScript)**
  ```
   ibmmfpfanalytics.logger.updateConfigFromServer();
  ```

What to do next

You can configure the client log levels from the MobileFirst Operations Console. For more information, see "Setting Log Filters from the MobileFirst Operations Console" on page 11-24.

Analytics workflows

Leverage MobileFirst Analytics to best serve your business needs. Once your goals are identified collect the appropriate data using the analytics client SDK and build reports using the MobileFirst Analytics Console.

These typical scenarios demonstrate methods of collecting and reporting analytics data.

App usage analytics

Collect data and build reports about app usage.

Initializing your app to capture app usage

App usage measures the number of times a specific app is brought to the foreground, and then sent to the background. To capture app usage in your mobile app, the MobileFirst Analytics client SDK must be configured to listen for the app lifecycle events.

About this task

You can use the MobileFirst Analytics API to capture app usage. Make sure you have first created a relevant device listener.

- On iOS, add the following code in your Application Delegate `application:didFinishLaunchingWithOptions` method.

  ```
  WLAnalytics *analytics = [WLAnalytics sharedInstance];
  [analytics addDeviceEventListener:LIFECYCLE];
  ```

- Similarly, on Android, add the following code in your Application subclass `onCreate` method.

  ```
  WLAnalytics.init(this)
  WLAnalytics.addDeviceEventListener(DeviceEvent.LIFECYCLE);
  ```

- For Cordova apps, the listener must be created in the native code. See Initilization for Android Cordova apps and Initialization for iOS Cordova apps.
- For Web apps, no listeners are required. Analytics can be enabled and disabled through the `WLlogger` class.

  ```
  logger.config({analyticsCapture: true});
  ```

Default Usage and Devices charts

Default charts are displayed in the IBM MobileFirst Analytics Console.

In the **Usage and Devices** page of the **Apps** section in the IBM MobileFirst Analytics Console, a number of default charts are provided to help you manage your app usage.

Total Devices

The **Total Devices** chart shows the number of total devices.

Total App Sessions

The **Total App Sessions** chart shows the number of total app sessions. An app session is recorded when an app is brought to the foreground of a device.

Active Users

The **Active Users** chart shows an interactive multi-line graph of the following data:

- **Active Users** - unique users for the displayed time frame.
- **New Users** - new users for the displayed time frame.

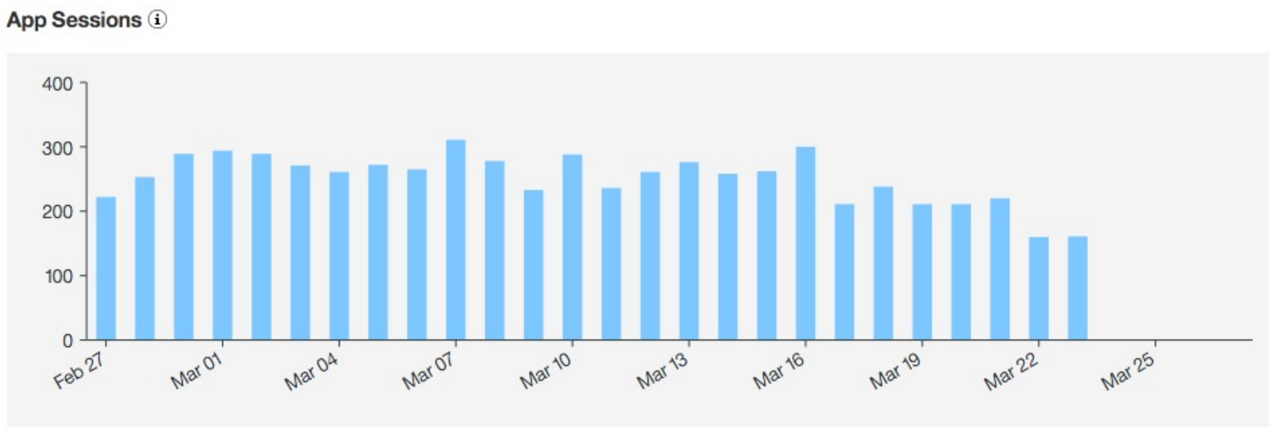
The default displayed time frame is one day with a data point for each hour. If you change the displayed time frame to greater than one day, the data points reflect each day. You can click the corresponding key in the legend to toggle whether to display the line. By default, all keys are displayed, and you cannot toggle all keys to not display any lines.

To see the most accurate data in the line graph, you must instrument your app code to provide the `userID` by calling the `setUserContext(WLAnalytics)` API. If you want to provide anonymity for the `userID` values, you must hash the value first. If the `userID` is not provided, the ID of the device is used by default. If one device is used by multiple users and the `userID` is not provided, the line graph does not reflect accurate data because the ID of the device counts as one user.

App Sessions

The **App Sessions** chart shows a bar graph of app sessions over time.

The following image shows a sample **App Sessions** chart.



For more information about how to populate this chart, see "Initializing your app to capture app usage" on page 11-42.

App Usage

The **App Usage** chart shows a pie chart of the percentage of app sessions for each app.

New Devices

The **New Devices** chart shows a bar graph of new devices over time.

Model Usage

The **Model Usage** chart shows a pie chart of the percentage of app sessions for each device model.

Operating System Usage

The **Operating System Usage** chart shows a pie chart of the percentage of app sessions for each device operating system.

Creating a custom chart for average session duration

The duration of an app session is a valuable metric to visualize. With any app, you want to know the amount of time that users are spending on a particular session.

About this task

You can create a custom chart to view this type of information. To view average session duration in a custom chart, follow these steps.

Procedure

1. Ensure that you are collecting app sessions as described in "Initializing your app to capture app usage" on page 11-42.

2. In the MobileFirst Analytics Console, click **Create Chart** in the **Custom Charts** page of the **Dashboard** section.
3. Give your chart a title.
4. Select **App Session** for **Event Type**.
5. Select **Bar Graph** for **Chart Type**.
6. Click **Next**.
7. Select **Timeline** for **X-Axis**.
8. Select **Average** for **Y-Axis**.
9. Select **Duration** for **Property**.
10. Click **Save**.

Results

A bar graph is created that displays average app session duration over time. You can enhance this chart by adding a threshold or chart filters. For more information, see "Custom charts" on page 11-25.

Crash capture

Capture data about application crashes and create relevant reports.

MobileFirst Analytics includes data and reports about application crashes. This data is collected automatically along with other lifecycle event data. The crash data is collected by the client and is sent to the server once the application is again up and running.

Initializing your app to capture crash data

An app crash is recorded when an unhandled exception occurs and causes the program to be in an unrecoverable state. Before the app closes, the MobileFirst Analytics SDK logs a crash event. This data is sent to the server with the next logger `send` call.

About this task

To ensure that crash data is collected and included in the MobileFirst Analytics Console reports, make sure the crash data is sent to the server.

Procedure

1. Ensure that you are collecting app lifecycle events as described in "Initializing your app to capture app usage" on page 11-42.
2. The client logs must be sent once the app is running again, in order to get the stacktrace that is associated with the crash.
   - **iOS**

```
- (void)sendMFPAnalyticData {
  [OCLogger send];
  [[WLAnalytics sharedInstance] send];
}

// then elsewhere in the same implementation file:

[NSTimer scheduledTimerWithTimeInterval:60
  target:self
  selector:@selector(sendMFPAnalyticData)
  userInfo:nil
  repeats:YES]
```

- **Android**

  ```
  Timer timer = new Timer();
  timer.schedule(new TimerTask() {
    @Override
    public void run() {
      Logger.send();
      WLAnalytics.send();
    }
  }, 0, 60000);
  ```
- **Cordova**

  ```
  setInterval(function() {
    WL.Logger.send();
    WL.Analytics.send();
  }, 60000)
  ```
- **web**

  ```
  setInterval(function() {
    ibmmfpfanalytics.logger.send();
  }, 60000);
  ```

App crash monitoring

You can quickly see information about your app crashes in the **Dashboard** section of the MobileFirst Analytics Console.

In the **Overview** page of the **Dashboard** section, 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: The **Crashes** chart queries against the `MfpAppSession` documents. You must instrument your app 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 chart 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 limitation:
- The **Crashes** bar graph displays no data when **Display Crash Rate** is selected.

To instrument crash data, see "Initializing your app to capture crash data" on page 11-45.

App crash troubleshooting

You can view the **Crashes** page in the **Applications** section of the MobileFirst Analytics Console to better administer your apps.

The **Crash Overview** table shows the following data columns:
- **App**: app name
- **Crashes**: total number of crashes for that app
- **Total Uses**: total number of times a user opens and closes that app
- **Crash Rate**: percentage of crashes per use

The **Crashes** bar graph is the same chart that is displayed in the **Overview** page of the **Dashboard** section.

Note: Both charts query against the `MfpAppSession` documents. You must instrument your app 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 "Initializing your app to capture crash data" on page 11-45.

The **Crash Summary** table is sortable and includes the following data columns:

- **Crashes**
- **Devices**
- **Last Crash**
- **App**
- **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**
- **App 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** table is populated by querying the fatal level client logs. If your app does not collect fatal client logs, no data is available.

Default charts for crashes

Default charts for crashes are displayed in the IBM MobileFirst Analytics Console.

In the **Crashes** page of the **Apps** section in the IBM MobileFirst Analytics Console, a number of default charts are provided to help you manage your app crashes.

Crash Overview

The **Crash Overview** chart shows a table of an overview of crashes.

For more information about the **Crash Overview** table, see "App crash troubleshooting" on page 11-46.

Crashes

The **Crashes** shows a bar graph histogram of crashes over time. You can display the data by crash rate or total crashes. The **Crashes** bar graph is also in the **Crashes** page of the **Applications** section.

For more information about the **Crashes** bar graph, see "App crash monitoring" on page 11-46.

Crash Summary

The **Crash Summary** chart shows a sortable table of a summary of crashes. You can expand the individual crashes by clicking the **+** icon to view a **Crash Details** table that includes more details about the crashes. In the **Crash Details** table, you can click the **>** icon to view more details about the specific crash instance.

Custom analytics

Capture and visualize custom data.

Instrumenting your app to capture custom analytics

Custom analytics can be captured by using the WLAnalytics API. The MobileFirst Analytics API provides a `log` method with which arbitrary key/value pairs can be recorded and sent to the MobileFirst Analytics Server.

About this task

Custom analytics can include any data you collect. For example:

- Page flow/transitions
- Recording gestures
- Recording button clicks

As an example to collect page transitions, follow these steps.

- On iOS, add the following code in the `ViewController`'s `viewDidLoad` method.

  ```
  NSArray *viewControllers = self.navigationController.viewControllers;
  NSUInteger numViewControllers = viewControllers.count;
  if (numViewControllers >= 2) {
    UIViewController *previous = [viewControllers objectAtIndex:(numViewControllers - 2)];
    NSDictionary *metadata = @{@"fromPage": previous.title, @"toPage": self.title};
    [[WLAnalytics sharedInstance] log:@"page transition" withMetadata:metadata];
  }
  ```

- On Android, add the following code in an `Activity` subclass' `onResume()` method.

  ```
  String fromActivity = getIntent().getStringExtra("fromActivity");
  JSONObject metadata = new JSONObject();
  metadata.put("fromPage", fromActivity);
  metadata.put("toPage", "This Page");
  WLAnalytics.log("page transition", metadata);
  ```

- In JavaScript (Cordova), add the following code.

  ```
  // before showing the element identified by 'setting'
  $(document).on('pagebeforeshow', '#setting', function(e, obj) {
    WL.Analytics.log('page transition', {
      'fromPage': obj.prevPage[0].id,
      'toPage': obj.toPage[0].id
    });
  });
  ```

- For the web API, custom data is sent by the `addEvent` method.

  ```
  webAnalytics.addEvent({'Purchases':'radio'});
  webAnalytics.addEvent({'src':'App landing page','target':'About page'});
  ```

What to do next

As with all analytic data that is collected from the mobile app, the `send()` method must be called for the data to appear in the MobileFirst Analytics Console.

Visualizing custom analytics

Custom analytics can be visualized by creating custom charts in the MobileFirst Analytics Console.

About this task

To visualize the page transitions with a flow chart, follow these steps.

Procedure

1. Ensure that you instrumented your app to capture custom analytics as described in "Instrumenting your app to capture custom analytics" on page 11-48.
2. In the MobileFirst Analytics Console, click **Create Chart** in the **Custom Charts** page of the **Dashboard** section.
3. Give your chart a title.
4. Select **Custom Data** for **Event Type**.
5. Select **Flow Chart** for **Chart Type**.
6. Click **Next**.
7. Select **fromPage** for **Source**.
8. Select **toPage** for **Destination**.
9. Select the page for which you want to visualize the page flow for **Property**.
10. Click **Save**.

Troubleshooting Analytics and Logger

Find information to help resolve analytics and logger problems that you might encounter.

Quick Links

- "Why is there no data?"
- "Why is there crash data in the Crash Overview table, but nothing in the Crash Summary table?" on page 11-50
- "Why is there no data in the Server Usage Flow graph or the Network Request graph?" on page 11-50
- "Why is there no data for app sessions?" on page 11-50

Why is there no data?

Check the following possibilities.

- Verify that your apps are set to point to the MobileFirst Server, which forwards the logs to the MobileFirst Analytics Server. Ensure that the following values are set in the `mfpclient.plist` (iOS), `mfpclient.properties` (Android), or `config.xml` (Cordova) files.
  - `protocol` = http
  - `host` = the IP address of your MobileFirst Server
  - `port` = the HTTP port that is set in the `server.xml` file for reporting analytics

  - `wlServerContext` = /mfp/
- Ensure that your MobileFirst Server is pointing to your MobileFirst Analytics Server.
  - `/analytics-service`
  - `/analytics`
- Check that you are calling the `send` method.
  - iOS: `[[WLAnalytics sharedInstance] send];`
  - Android: `WLAnalytics.send();`
  - Cordova: `WL.Analytics.send();`

Why is there crash data in the Crash Overview table, but nothing in the Crash Summary table?

Verify that your apps are sending logs after a crash. To be safe, send logs on app start-up to ensure that any previously unsent information is reported.

Why is there no data in the Server Usage Flow graph or the Network Request graph?

Configure your apps to collect analytics on the `Network` device event.

- For cross-platform apps that use Cordova, follow the iOS or Android guides, as the configurations are the same as for native apps.
- To enable the capture of network analytic data in iOS, add the following code in your Application Delegate `application:didFinishLaunchingWithOptions` method.
  ```
  WLAnalytics *analytics = [WLAnalytics sharedInstance];
  [analytics addDeviceEventListener:NETWORK];
  ```
- To enable the capture of network analytic data in Android, add the following code in your Application subclass `onCreate` method.
  ```
  WLAnalytics.init(this);
  WLAnalytics.addDeviceEventListener(DeviceEvent.NETWORK);
  ```

Why is there no data for app sessions?

Configure your apps to collect analytics on the `Lifecycle` device event.

- For cross-platform apps that use Cordova, follow the iOS or Android guides, as the configurations are the same as for native apps.
- To enable the capture of network analytic data in iOS, add the following code in your Application Delegate `application:didFinishLaunchingWithOptions` method.
  ```
  WLAnalytics *analytics = [WLAnalytics sharedInstance];
  [analytics addDeviceEventListener:LIFECYCLE];
  ```
- To enable the capture of network analytic data in Android, add the following code in your Application subclass `onCreate` method.
  ```
  WLAnalytics.init(this);
  WLAnalytics.addDeviceEventListener(DeviceEvent.LIFECYCLE);;
  ```

Integrating with other IBM products

IBM MobileFirst Platform Foundation integrates with other IBM products.

You can find samples and documentation about such integration for developers and administrators on the Product Integration page of the Developer Center website for IBM MobileFirst Platform.

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 or the Android Market, 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.

In the current version, Application Center supports applications that are built for the Google Android platform, the Apple iOS platform, the Windows Phone 8 platform, and the Windows 8 platform.

For Windows Phone, only the Windows Phone application package (`.xap`) file format is currently supported, **not** the app package (`.appx`) file format (universal app format). For Windows Store (Desktop applications), the app package (`.appx`) file format is supported.

Windows Phone 7 and Windows RT, and BlackBerry OS are not supported by the current version of the Application Center.

Application Center manages mobile applications; it supports any kind of Android, iOS, Windows Phone 8, or Windows 8 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 Android, iOS, Windows Phone, or Windows 8 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

Different operating systems impose specific requirements for deploying, installing, or using applications on the appropriate mobile devices.

Android
The mobile device must be configured for installation from unknown sources. The corresponding toggle can be found in the Android Settings. See User Opt-in for apps from unknown sources for details.

In Application Center, applications have an internal and a commercial version number. The internal version number is used to distinguish which version is newer while the commercial version is only used as an informative display string. For Android applications, the internal version is the android:versionCode from the application manifest, and it must be an integer.

iOS
All 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.

In Application Center, applications have an internal and a commercial version number. The internal version number is used to distinguish which version is newer while the commercial version is used only as an informative display string. For iOS applications, the internal version is the CFBundleVersion from the application manifest `Info.plist`. The version number must have the following format: `a`, or `a.b`, or `a.b.c`, where `a`, `b`, `c` are non-negative integers, and `a` is not `0`.

Windows Phone 8
Applications are not installed from the Windows Store, but from

Application Center, which acts as what Microsoft documentation calls a *Company Hub*. See Company app distribution for Windows Phone for details.

To use a company hub, Windows Phone requires you to register a company account with Microsoft and to sign all applications, including the Application Center client, with the company certificate. Only signed applications can be managed through Application Center.

You must enroll all mobile devices through an application enrollment token that is associated with your company account.Application Center helps you to enroll devices through facilities to distribute the application enrollment token. See "Application enrollment tokens in Windows 8 Universal" on page 13-44 for details.

Application Center supports the distribution of applications as Windows Phone application package (`.xap`) files for Microsoft Windows Phone 8.0 and Microsoft Windows Phone 8.1. With Microsoft Windows Phone 8.1, Microsoft introduced a new universal format as app package (`.appx`) files for Windows Phone. Currently, Application Center does not support the distribution of app package (`.appx`) files for Microsoft Windows Phone 8.1, but is limited to Windows Phone application package (`.xap`) files only.

In Application Center, applications have only one version number. The version number is used to distinguish which version is newer. For Windows Phone 8 applications, the version number is in the `Version` field in the `WMAppManifest.xml` file. This version number must have the following format: `a.b.c.d` where `a`, `b`, `c`, `d` are non-negative integers.

Windows 8

The Application Center mobile client is provided as a normal desktop executable file (`.exe`). Use it to install on the device Windows Store applications, which are packaged as `.appx` files.

Installing a file of type `appx` on your device without using Windows Store is called *sideloading* an app. To sideload an app, you must comply with the prerequisites in Prepare to sideload apps. The Windows 8.1 update simplifies the prerequisites for sideloading. For more information, see Sideloading store apps to Windows 8.1 devices.

Files of type `.exe` cannot be executed on ARM-based tablets, so Application Center does not support Windows RT; only Windows 8 and Windows 8.1 are supported.

The device user needs administrator rights on the device to execute the Application Center client.

Application Center does not provide any predefined way of distributing the mobile client.

In Application Center, applications have only one version number. The version number is used to distinguish which version is newer. For Windows 8 applications, the version number is in the `Version` field in the `AppxManifest.xml` file. This version number must have the following format: `a.b.c.d`, where `a`, `b`, `c`, `d` are non-negative integers.

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 13-18.

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 13-51.

The following figure shows an overview of the architecture.

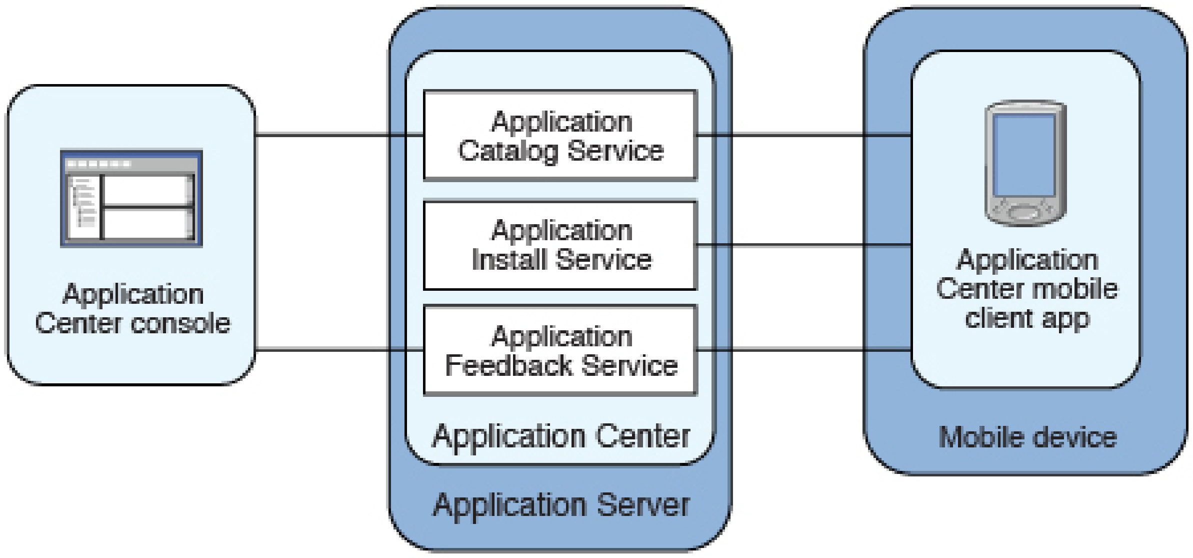

Application Center console
Application Catalog Service
Application Install Service
Application Feedback Service
Application Center
Application Server
Application Center mobile client app
Mobile device

Figure 13-1. Architecture of the Application Center

From the Application Center console, you can take the following actions:
- Upload different versions of mobile applications.
- Remove unwanted applications.
- Control access to applications: 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 take the following actions:
- List available mobile applications.
- Install a new application on a device.
- Send feedback about an application.

The Application Center supports applications for Android, iOS, Windows Phone 8, and Windows 8 devices. Therefore, the mobile client comes in separate versions for Android, iOS, Windows Phone 8, and Windows 8.

The Android, iOS, and Windows Phone 8 mobile clients are built on the MobileFirst platform.To learn how to configure the Application Center server-side component on various Java application servers after the product is installed and build MobileFirst applications for the Application Center client, see “Configuring Application Center after installation” on page 6-233.

Preliminary information

To use the Application Center, you must configure security settings, start the web application server where IBM MobileFirst Platform Foundation is installed, start the Application Center console, and log in.

When you install IBM MobileFirst Platform Foundation, the Application Center is automatically installed in the specified application server.

If you install the Application Center in WebSphere Application Server Liberty profile, the server is created 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 6-233 or, if you are using LDAP authentication, "Managing users with LDAP" on page 6-237.

Example: starting the server and the Application Center console on Liberty profile

1. Start the Liberty server by using the `server` command that is in the *installation-directory*/server/wlp/bin directory.
   ```
   server start worklightServer
   ```
2. When the server is running, start the Application Center console by entering this address in your browser: `http://localhost:9080/appcenterconsole/`
3. Log in. By default, two users are defined for the installation of the Application Center on Apache Tomcat or WebSphere Application Server Liberty profile:
   - **demo** with password `demo`
   - **appcenteradmin** with password `admin`

For more information

To use the Application Center console, refer to "The Application Center console" on page 13-18.

To install and run the mobile client on the following operating systems, see:

- Android: See "Installing the client on an Android mobile device" on page 13-51
- iOS operating system: See "Installing the client on an iOS mobile device" on page 13-54.
- Windows Phone 8: See "Installing the client on Windows 8 Universal" on page 13-58.
- Windows 8: The mobile client for Windows 8 is not intended to be deployed in Application Center for later distribution. See "Microsoft Windows 8: Building the project" on page 13-11.

Preparations for using the mobile client

To use the mobile client to install apps on mobile devices, you must either generate the app by using the provided Eclipse and Visual Studio projects or use the version of the client provided for Android, iOS, or Windows 8 Universal, directly.

Prerequisites for building the Application Center installer

The Application Center comes with an Android, an iOS, and Windows 8 Universal 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 Android, the iOS, and the Windows 8 Universal versions of the client.

The Windows 8 Universal project is provided as a Visual Studio project located at IBMApplicationCenterWindowsStore\AppCenterClientWindowsStore.csproj.

Prerequisites specific to the Android operating system

The Android version of the mobile client is included in the software delivery in the form of an Android application package (.apk) file. The IBMApplicationCenter.apk file is in the directory ApplicationCenter/installer. Push notifications are disabled. If you want to enable push notifications, you must rebuild the .apk file. See "Push notifications of application updates" on page 13-12 for more information about push notifications in the Application Center.

To build the Android version, you must have the latest version of the Android development tools.

Prerequisites specific to Apple iOS operating system

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 Studio 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.

Note: For V8.0.0, use MobileFirst Studio 7.1. You can download MobileFirst Studio from the Downloads page of the Developer Center website. Click the **Previous MobileFirst Platform Foundation releases** tab for the download link. For installation instructions, see Installing MobileFirst Studio in the IBM Knowledge Center for 7.1.

Prerequisites specific to Microsoft Windows Phone operating system

The Windows Phone version of the mobile client is included as an unsigned Windows Phone application package (.xap) file in the software delivery. The IBMApplicationCenterUnsigned.xap file is in the ApplicationCenter/installer directory.

Important: The unsigned .xap file cannot be used directly. You must sign it with your company certificate obtained from Symantec/Microsoft **before** you can install it on a device.

Optional: If necessary, you can also build the Windows Phone version from sources. For this purpose, you must have the latest version of Microsoft Visual Studio.

Prerequisites specific to Microsoft Windows 8 operating system

The Windows 8 version of the mobile client is included as a .zip archive file. The IBMApplicationCenterWindowsStore.zip file contains an executable file (.exe) and its dependent Dynamic-Link Library (.dll) files. To use the content of this archive, you download the archive to a location on you local drive and run the executable file.

Optional: If necessary, you can also build the Windows 8 version from sources. For this purpose, you must have the latest version of Microsoft Visual Studio.

Importing and building the project (Android, iOS, Windows Phone)

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, but if you purchased this product, you are entitled to the full cross-platform version of the product as well.

Note: For V8.0.0, use MobileFirst Studio 7.1. You can download MobileFirst Studio from the Downloads page of the Developer Center website. Click the **Previous MobileFirst Platform Foundation releases** tab for the download link. For installation instructions, see Installing MobileFirst Studio in the IBM Knowledge Center for 7.1.

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**.

Android

MobileFirst Studio generates a native Android project in `IBMAppCenter/apps/AppCenter/android/native`. A native Android development tools (ADT) project is in the `android/native` folder. You can compile and sign this project by using the ADT tools. This project requires Android SDK level 16 to be installed, so that the resulting APK is compatible with all Android versions 2.3 and later. If you choose a higher level of the Android SDK when you build the project, the resulting APK will not be compatible with Android version 2.3.

See the Android site for developers for more specific Android information that affects the mobile client application.

If you want to enable push notifications for application updates, you must first configure the Application Center client properties. See "Configuring push notifications for application updates" on page 13-13 for more information.

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 13-13 for more information.

Windows Phone 8
MobileFirst Studio generates a native Windows Phone 8 project in `IBMAppCenter/apps/AppCenter/windowsphone8/native`. The `AppCenter.csproj` file is in the `windowsphone8/native` folder. This file is the Visual Studio project that you must compile by using Visual Studio and the Windows Phone 8.0 SDK.

The application is built with the Windows Phone 8.0 SDK so that it can run on Windows Phone 8.0 and 8.1 devices. It is not built with the Windows Phone 8.1 SDK, because the result would not run on earlier Windows Phone 8.0 devices.

The installation of Visual Studio 2013 enables you to select the installation of the Windows Phone 8.0 SDK in addition to the 8.1 SDK. The Windows Phone 8.0 SDK is also available from Windows Phone SDK Archives.

See Windows Phone Dev Center to learn more about how to build and sign the Windows Phone mobile client application.

Customizing features (for experts): Android, iOS, Windows Phone

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 13-1. 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.
gcmProjectId	The Google API project ID (project name = `com.ibm.appcenter`), which is required for Android push notifications; for example, 123456789012.
allowAppLinkReview	This property indicates whether local reviews of applications from external application stores can be registered and browsed in the Application Center. These local reviews are not visible in the external application store. These reviews are stored in the Application Center server.

Other resources

Other resources that are available are application icons, application name, splash screen images, icons, and translatable resources of the application.

Application icons

Android: The file named `icon.png` in the `IBMAppCenter/apps/AppCenter/android/native/res/drawable`*`density`* directories; one directory exists for each density.

iOS: Files named `icon`*`size`*`.png` in the `IBMAppCenter/apps/AppCenter/iphone/native/Resources` directory.

Windows Phone: Files named `ApplicationIcon.png`, `IconicTileSmallIcon.png`, and `IconicTileMediumIcon.png` in the `IBMAppCenter/apps/AppCenter/windowsphone8/native` directory.

Application name

Android: Edit the **`app_name`** property in the `IBMAppCenter/apps/AppCenter/android/native/res/values/strings.xml` file.

iOS: Edit the **`CFBundleDisplayName`** key in the `IBMAppCenter/apps/AppCenter/iphone/native/IBMAppCenterAppCenterIphone-Info.plist` file.

Windows Phone: Edit the **`Title`** attribute of the **`App`** entry in the `IBMAppCenter/apps/AppCenter/windowsphone8/native/Properties/WMAppManifest.xml` file.

Splash screen images

Android: Edit the file named `splashimage.9.png` in the `IBMAppCenter/apps/AppCenter/android/native/res/drawable/`*`density`* directories; one directory exists for each density. This file is a patch 9 image.

iOS: Files named `Default-`*`size`*`.png` in the `IBMAppCenter/apps/AppCenter/iphone/native/Resources` directory.

Hybrid splash screen during auto login: `/IBMAppCenter/apps/AppCenter/common/js/idx/mobile/themes/common/idx/Launch.css`

Windows Phone: Edit the file named SplashScreenImage.png in the `IBMAppCenter/apps/AppCenter/windowsphone8/native` directory.

Icons (buttons, stars, and similar objects) of the application

`IBMAppCenter/apps/AppCenter/common/css/images`.

Translatable resources of the application

`IBMAppCenter/apps/AppCenter/common/js/appcenter/nls/common.js`.

Microsoft Windows 8: Building the project

Build the Application Center client project for Windows 8 in Microsoft Visual Studio 2013.

About this task

You must build the client project in Microsoft Visual Studio 2013 before you can distribute it.

Building the project is a prerequisite to distributing it to your users, but the Windows 8 mobile client is not intended to be deployed on Application Center for later distribution.

Procedure

To build the Windows 8 project:

1. Open the Visual Studio project file called `IBMApplicationCenterWindowsStore\AppCenterClientWindowsStore.csproj` in Microsoft Visual Studio 2013.
2. Perform a full build of the application.

What to do next

To distribute the mobile client to your Application Center users, you can later generate an installer that will install the generated executable (.exe) file and its dependent Dynamic-Link Library (.dll) files. Alternatively, you can provide these files without including them in an installer.

Deploying the mobile client in Application Center

Deploy the different versions of the client application to Application Center.

The Windows 8 mobile client is not intended to be deployed in Application Center for later distribution. You can choose to distribute the Windows 8 mobile client either by providing users with the client `.exe` executable file and dynamic link library `.dll` files directly packaged in an archive, or by creating an executable installer for the Windows 8 mobile client.

The Android, iOS, and Windows Phone versions of the mobile client must be deployed to the Application Center. To do so, you must upload the Android application package (`.apk`) files, iOS application (`.ipa`) files, and Windows Phone application (`.xap`) files, Web directory archive (`.zip`) files to the Application Center.

Follow the steps described in "Adding a mobile application" on page 13-22 to add the mobile client application for Android, iOS, and Windows Phone. Make sure that you select the **`Installer`** application property to indicate that the application is an installer. Selecting this property enables mobile device users to install the mobile client application easily over the air. To install the mobile client, see the related task that corresponds to the version of the mobile client app determined by the operating system.

Related tasks:

"Installing the client on an Android mobile device" on page 13-51
You can install the mobile client, or any signed application marked with the installer flag, on your Android mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

"Installing the client on an iOS mobile device" on page 13-54
You can install the mobile client, or any signed application marked with the installer flag, on your iOS mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

"Installing the client on Windows 8 Universal" on page 13-58
You can install the mobile client, or any signed application marked with the installer flag, on Windows 8 Universal by entering the access URL in your browser, entering your credentials, and completing the required steps. The company account must be preinstalled on your mobile device.

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 send notification automatically, to any iOS or Android 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.

  iOS and modern Android operating system versions offer a way to switch this service on or off on a per application basis.

  Refer to your device vendor to learn how to configure your mobile device for push notifications.

Related concepts:

"Marking or unmarking a favorite app" on page 13-85
Mark your favorite apps or unmark an app to have it removed from the favorites list.

Related reference:

"Application properties" on page 13-27
Applications have their own sets of properties, which depend on the operating system on the mobile device and 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 Google or Apple push notification servers.

Purpose

You must configure the credentials or certificates of 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 `ibm.appcenter.push.schedule.period.amount` and `ibm.appcenter.push.schedule.period.unit` server environment variables.

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, avoid setting the unit to `seconds`. The shorter the elapsed time, the higher the load on the server. The unit expressed in seconds is implemented only for testing and evaluation purposes. For example, when the elapsed time is set to 10 seconds, push messages are sent almost immediately.

See "JNDI properties for Application Center" on page 6-261 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" value="hours"/>
<Environment name="ibm.appcenter.push.schedule.period.amount" override="false" type="java.lang.String" value="2"/>
```

WebSphere Application Server v8.5

To configure JNDI variables for WebSphere Application Server v8.5, proceed as follows:

1. Click **Applications** > **Application Types** > **Websphere enterprise applications**.
2. Select the Application Center Services application.
3. Click **Web Module Properties** > **Environment entries for Web modules**.
4. Edit the string in the **Value** column.

WebSphere Application Server Liberty profile

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.

Configuring the Application Center server for connection to Google Cloud Messaging

Enable Google Cloud Messaging (GCM) for your application.

About this task

To enable Google Cloud Messaging (GCM) for an application, you must attach the GCM services to a developer Google account with the Google API enabled. See Getting Started with GCM for details.

Important: The Application Center client without Google Cloud Messaging: The Application Center relies on the availability of the Google Cloud Messaging (GCM) API. This API might not be available on devices in some territories such as China. To support those territories, you can build a version of the Application Center client that does not depend on the GCM API. The push notification feature does not work on that version of the Application Center client. See "Building a version of the mobile client that does not depend on the GCM API" on page 13-18 for details.

Procedure

1. If you do not have the appropriate Google account, go to Create a Google account and create one for the Application Center client.
2. Register this account by using the Google API in the Google API console. Registration creates a new default project that you can rename. The name you give to this GCM project is not related to your Android application package name. When the project is created, a GCM project ID is appended to the end of the project URL. You should record this trailing number as your project ID for future reference.
3. Enable the GCM service for your project; in the Google API console, click the **Services** tab on the left and enable the "Google Cloud Messaging for Android" service in the list of services.
4. Make sure that a Simple API Access Server key is available for your application communications.
   a. Click the **API Access** vertical tab on the left of the console.
   b. Create a Simple API Access Server key or, if a default key is already created, note the details of the default key. Two other kinds of key exist that are not of interest at this time.
   c. Save the Simple API Access Server key for future use in your application communications through GCM. The key is about 40 characters long and is referred to as the Google API key that you will need later on the server side.
5. Enter the GCM project ID as a string resource property in the JavaScript project of the Application Center Android client; in the `IBMAppCenter/apps/AppCenter/common/js/appcenter/config.json` template file, modify this line with your own value:

```
gcmProjectId:""// Google API project (project name = com.ibm.appcenter) ID needed for Android push.
// example : 123456789012
```

6. Register the Google API key as a JNDI property for the Application Center server. The key name is : **`ibm.appcenter.gcm.signature.googleapikey`**. For example, you can configure this key for an Apache Tomcat server as a JNDI property in the `server.xml` file:

```
<Context docBase="AppCenterServices" path="/applicationcenter" reloadable="true" source="org.eclipse.jst.jee.server:AppCenterServices">
<Environment name="ibm.appcenter.gcm.signature.googleapikey" override="false" type="java.lang.String"
value="AIxaScCHg0VSGdgfOZKtzDJ44-oi0muUasMZvAs"/>
</Context>
```

   The JNDI property must be defined in accordance with your application server requirements.

   See "JNDI properties for Application Center" on page 6-261 for a complete list of properties that you can set.

   Important:

   - If you use GCM with earlier versions of Android, you might need to pair your device with an existing Google account for GCM to work effectively. See GCM service: "It uses an existing connection for Google services. For pre-3.0 devices, this requires users to set up their Google account on their mobile devices. A Google account is not a requirement on devices running Android 4.0.4 or higher."

- You must also ensure that your firewall accepts outgoing connections to `android.googleapis.com` on port 443 for push notifications to work.

Configuring the Application Center server for connection to Apple Push Notification Services

Configure your iOS project for Apple Push Notification Services (APNs).

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.

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 13-2. 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.location`*`" override="false" type="java.lang.String" value= "/Users/someUser/someDirectory/apache-tomcat/conf/AppCenter_apns_dev_cert.p12"/>`
`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.password`*`" override="false" 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.isDevelopmentCertificate" override="false" type="java.lang.String" value="true"/>`

See “JNDI properties for Application Center” on page 6-261 for a complete list of JNDI properties that you can set.

Building a version of the mobile client that does not depend on the GCM API

You can remove the dependency on Google Cloud Messaging (GCM) API from the Android version of the client to comply with constraints in some territories. Push notifications do not work on this version of the client.

About this task

The Application Center relies on the availability of the Google Cloud Messaging (GCM) API. This API might not be available on devices in some territories such as China. To support those territories, you can build a version of the Application Center client that does not depend on the GCM API. The push notification feature does not work on that version of the Application Center client.

Procedure

1. Check that push notifications are disabled by checking that the `IBMAppCenter/apps/AppCenter/common/js/appcenter/config.json` file contains this line: `"gcmProjectId": "" ,`.
2. Remove from two places in the `IBMAppCenter/apps/AppCenter/android/native/AndroidManifest.xml` file all the lines that are located between these comments: `<!-- AppCenter Push configuration -->` and `<!-- end of AppCenter Push configuration -->`.
3. Delete the IBMAppCenter/apps/AppCenter/android/native/src/com/ibm/appcenter/GCMIntenteService.java class.
4. In Eclipse, run "Build Android Environment" in the `IBMAppCenter/apps/AppCenter/android` folder.
5. Delete the `IBMAppCenter/apps/AppCenter/android/native/libs/gcm.jar` file that was created by the MobileFirst plug-in when you ran the previous "Build Android Environment" command.
6. Refresh the newly created `IBMAppCenterAppCenterAndroid` project, so that the removal of the GCM library is taken into account.
7. Build the .apk file of the Application Center.

What to do next

The `gcm.jar` library is automatically added by the MobileFirst Eclipse plug-in each time that the Android environment is built. Therefore, this java archive file must be deleted from the `IBMAppCenter/apps/AppCenter/android/native/libs/` directory each time that the MobileFirst Android build process is run. Otherwise, the `gcm.jar` library is present in the resulting `appcenter.apk` file.

The Application Center console

With the Application Center console, you can manage the repository of the Application Center and your applications.

The Application Center console is a web application to manage the repository of the Application Center. The Application Center repository is the central location where you store the mobile applications that can be installed on mobile devices.

Use the Application Center console to:

- Upload applications that are written for these operating systems: Android, iOS, Windows 8 (Windows Store packages only), or Windows Phone 8.

- Manage several different versions of mobile applications.
- Review the feedback of testers of mobile applications.
- Define the users who have the rights to list and install an application on the mobile devices.
- Track which applications are installed on which devices.

Note:

Only users with the administrator role can log in to the Application Center console.

Multicultural support: the user interface of the Application Center console has not been translated.

Starting the Application Center console

You can start the Application Center with your web browser and log in if you have the administrator role.

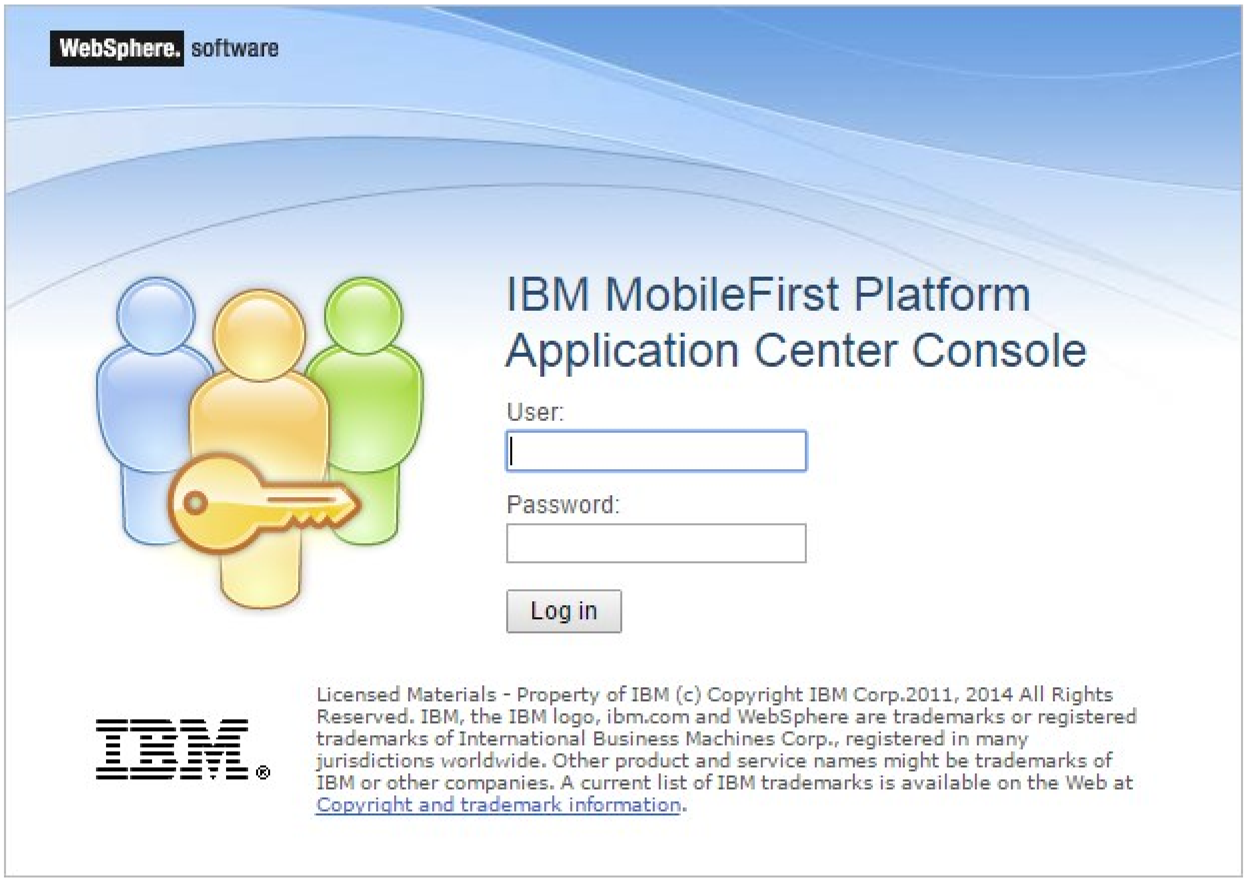
Procedure

1. Start a web browser session on your desktop.
2. Contact your system administrator to obtain the address and port of the server where the Application Center is installed.
3. Enter the following URL: `http://`*`server`*`/appcenterconsole`

   Where *server* is the address and port of the server where the Application Center is installed.

   `http://localhost:9080/appcenterconsole`
4. Log in to the Application Center console

   Contact your system administrator to get your credentials so that you can log in to the Application Center console.



Licensed Materials - Property of IBM (c) Copyright IBM Corp.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 13-2. Login of the Application Center console

Note:

Only users with the administrator role can log in to the Application Center console.

Troubleshooting a corrupted login page (Apache Tomcat)

You can recover from a corrupted 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 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.

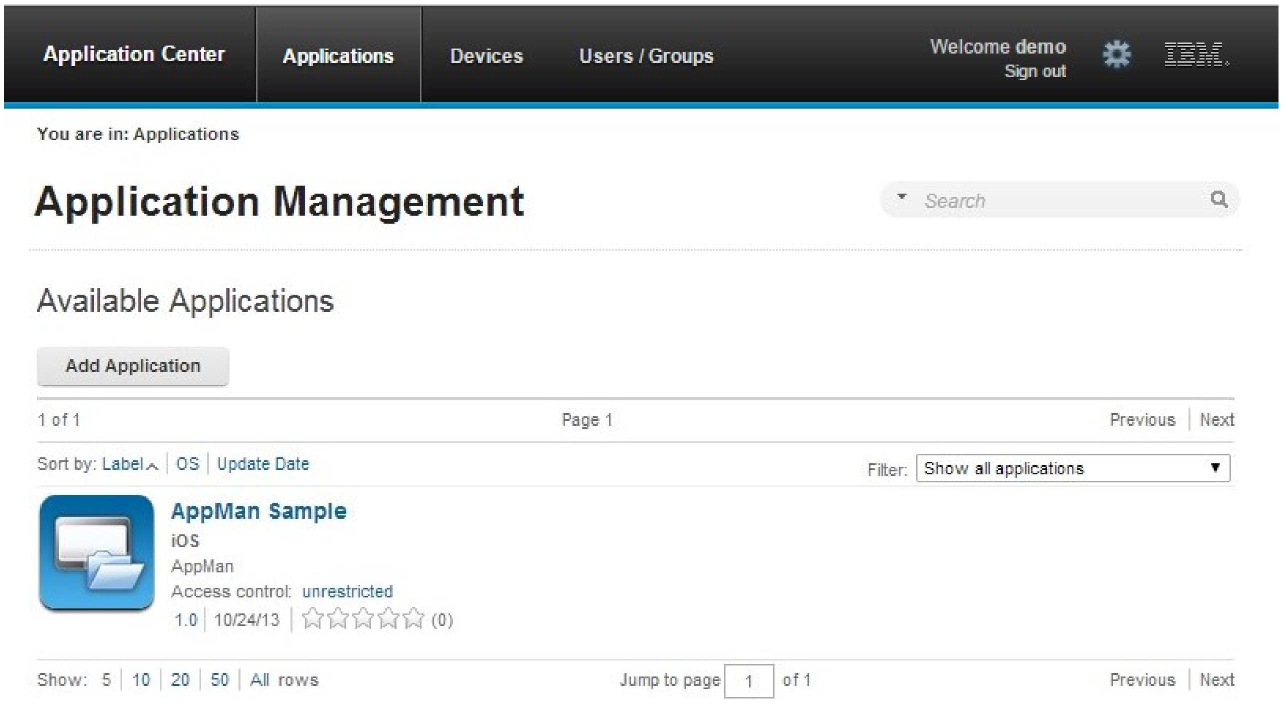


Figure 13-3. Available applications

Adding a mobile application

You can 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 for installation on mobile devices:

1. Click **Add Application**.
2. Click **Upload**.
3. Select the application file to upload to the Application Center repository.

   Android

   The application file name extension is `.apk`.

   iOS

   The application file name extension is `.ipa` for normal iOS applications.

Windows Phone 8

The application file name extension is `.xap`. The application must be signed with a company account. The application enrollment token for this company account must be made available to Windows Phone 8 devices before the application can be installed on the devices. See "Application enrollment tokens in Windows 8 Universal" on page 13-44 for details.

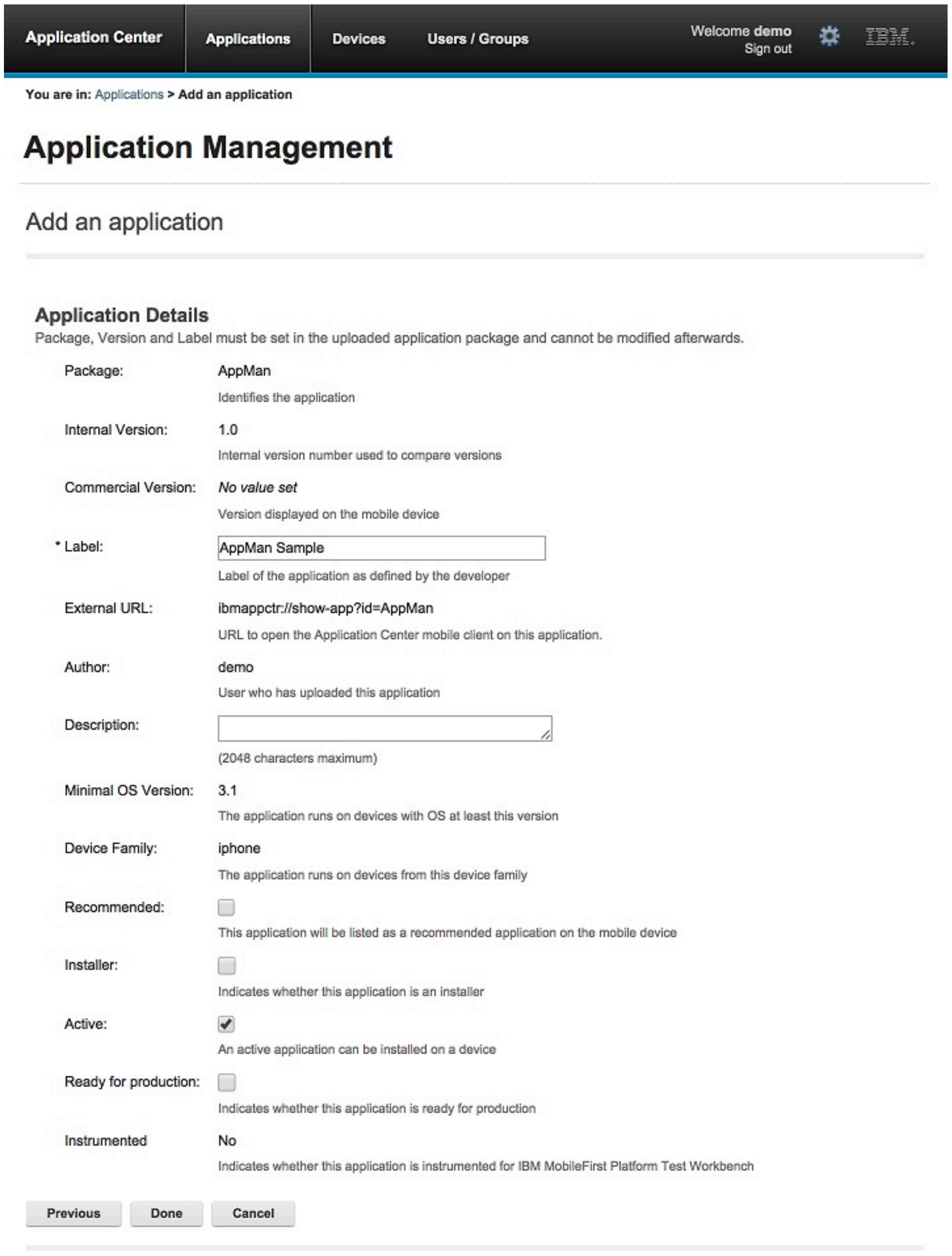
Windows 8

The application is provided as a Windows Store package; the file extension is `.appx`.

Windows Store `.appx` packages can depend on one or more Windows component library app packages, also known as "framework" packages. MobileFirst hybrid applications for Windows 8 depend on the Microsoft.WinJS framework package. When you use Microsoft Visual Studio to generate the application package, the dependencies packages are also generated and packaged as separate `.appx` files. To successfully install such applications by using the mobile client, you must upload the application `.appx` package and any other dependency package onto the Application Center server. When you upload a dependency package, it appears as inactive in the Application Center console. This behavior is expected, so that the framework package does not appear as an installable application in the client. Later, when a user installs an application, the mobile client checks whether the dependency is already installed on the device. If the dependency package is not installed, the client automatically retrieves the dependency package from the Application Center server and installs it on the device. For more information about dependencies, see Dependencies in the Windows developer documentation about packages and deployment of applications.

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**.

Results



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

Previous Done Cancel

Figure 13-4. 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 where you add an application that was created within your own enterprise. See "Adding a mobile application" on page 13-22.

Note: Currently, the Application Center supports only Google Play and Apple iTunes. Windows Phone Store and Windows Store are not yet supported.

Instead of the application executable file, you must provide a URL to the third-party application store where the application is stored. To find the correct application link more easily, the console provides direct links in the **Add an application** page to the supported third-party application store websites.

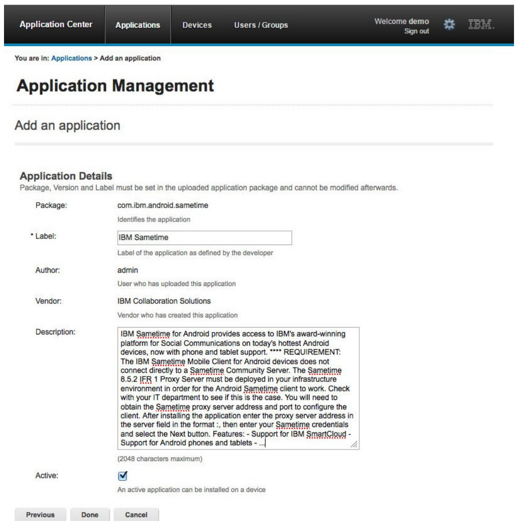
The Google play store address is https://play.google.com/store/apps.

The Apple iTunes store address is https://linkmaker.itunes.apple.com/; use the `linkmaker` site rather than the iTunes site, because you can search this site for all kinds of iTunes items, including songs, podcasts, and other items that are supported by Apple. Only selecting iOS applications provides you with compatible links to create application links.

Procedure

1. Click the URL of the public app store that you want to browse.
2. Copy the URL of the application in the third-party app store to the **Application URL** text field in the **Add an application** page of the Application Center console.
   - **Google Play**:
     a. Select an application in the store.
     b. Click the detail page of the application.
     c. Copy the address bar URL.
   - **Apple iTunes**:
     a. When the list of items is returned in the search result, select the item that you want.
     b. 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 unsuccessful, an error message is 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 is successful, this action displays the application properties. You can then modify the application description in the application properties before you move to the next step.



You are in: Applications > Add an application

Application Management

Add an application

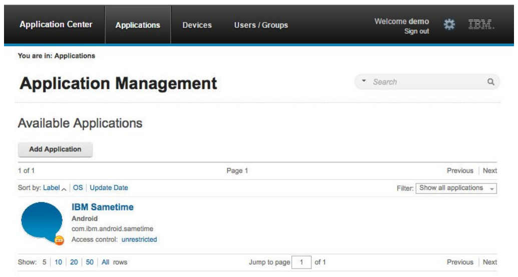
Application Details
Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package:	com.ibm.android.sametime Identifies the application
* Label:	IBM Sametime Label of the application as defined by the developer
Author:	admin User who has uploaded this application
Vendor:	IBM Collaboration Solutions Vendor who has created this application
Description:	IBM Sametime for Android provides access to IBM's award-winning platform for Social Communications on today's hottest Android devices, now with phone and tablet support. **** REQUIREMENT: The IBM Sametime Mobile Client for Android devices does not connect directly to a Sametime Community Server. The Sametime 8.5.2 IFR 1 Proxy Server must be deployed in your infrastructure environment in order for the Android Sametime client to work. Check with your IT department to see if this is the case. You will need to obtain the Sametime proxy server address and port to configure the client. After installing the application enter the proxy server address in the server field in the format :, then enter your Sametime credentials and select the Next button. Features: - Support for IBM SmartCloud - Support for Android phones and tablets - ... (2048 characters maximum)
Active:	✓ An active application can be installed on a device

Figure 13-5. Modified application description in application properties

4. Click **Done** to create the application link.

   This action makes the application available to the corresponding version of the Application Center mobile client. A small link icon appears on the application icon to show that this application is stored in a public app store and is different from a binary app.



You are in: Applications

Application Management

Available Applications

1 of 1 Page 1 Previous | Next

Sort by: Label | OS | Update Date Filter: Show all applications

IBM Sametime
Android
com.ibm.android.sametime
Access control: unrestricted

Show: 5 | 10 | 20 | 50 | All rows Jump to page 1 of 1 Previous | Next

Figure 13-6. Link to an application stored in Google play

Related concepts:

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

Related tasks:

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

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

"Installing applications through public app stores" on page 13-77
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, which depend on the operating system on the mobile device and cannot be edited. Applications also have a common property and editable properties.

The values of the following fields are taken from the application and you cannot edit them.

- **`Package`**.
- **`Internal Version`**.
- **`Commercial Version`**.
- **`Label`**.
- **`External URL`**; this property is supported for applications that run on Android, iOS, and Windows Phone 8.

Properties of Android applications

For more information about the following properties, see the Android SDK documentation.

- **`Package`** is the package name of the application; **`package`** attribute of the **`manifest`** element in the manifest file of the application.
- **`Internal Version`** is the internal version identification of the application; **`android:versionCode`** attribute of the **`manifest`** element in the manifest file of the application.
- **`Commercial Version`** is the published version of the application.
- **`Label`** is the label of the application; **`android:label`** attribute of the **`application`** element in the manifest file of the application.
- **`External URL`** is a URL that you can use to have the mobile client of the Application Center started automatically in the Details view of the latest version of the current application.

Properties of iOS applications

For more information about the following properties, see the iOS SDK documentation.

- **`Package`** is the company identifier and the product name; **`CFBundleIdentifier`** key.
- **`Internal Version`** is the build number of the application; **`CFBundleVersion`** key of the application.
- **`Commercial Version`** is the published version of the application.
- **`Label`** is the label of the application; **`CFBundleDisplayName`** key of the application.
- **`External URL`** is a URL that you can use to have the mobile client of the Application Center started automatically in the Details view of the latest version of the current application.

Properties of Windows Phone 8 applications

For more information about the following properties, see the Windows Phone documentation.

- **`Package`** is the product identifier of the application; **`ProductID`** attribute of the **`App`** element in the manifest file of the application.

- **Internal Version** is the version identification of the application; **Version** attribute of the **App** element in the manifest file of the application.
- **Commercial Version**, like **Internal Version**, is the version of the application.
- **Label** is the title of the application; **Title** attribute of the **App** element in the manifest file of the application.
- **Vendor** is the vendor who created the application; **Publisher** attribute of the **App** element in the manifest file of the application.
- **External URL** is a URL that you can use to have the mobile client of the Application Center started automatically in the Details view of the latest version of the current application.
- **Commercial Version**, like **Internal Version**, is the version of the application.

Properties of Windows Store applications

For more information about the following properties, see the Windows Store documentation about application development.

- **Package** is the product identifier of the application; **Package name** attribute in the manifest file of the application.
- **Internal Version** is the version identification of the application; **Version** attribute in the manifest file of the application.
- **Commercial Version**, like **Internal Version**, is the version of the application.
- **Label** is the title of the application; **Package display name** attribute in the manifest file of the application.
- **Vendor** is the vendor who created the application; **Publisher** attribute in the manifest file of the application.

Common property: Author

The **Author** field is read-only. It displays the **username** attribute 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 encourage users to install this application. Recommended applications appear as a special list 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 13-51.

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 is displayed on the device and 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 is ready to be deployed in a production environment and is therefore suitable to be managed by Tivoli Endpoint Manager through its application store. Applications for which this property is selected are the only ones that are flagged to Tivoli Endpoint Manager.

Editing application properties

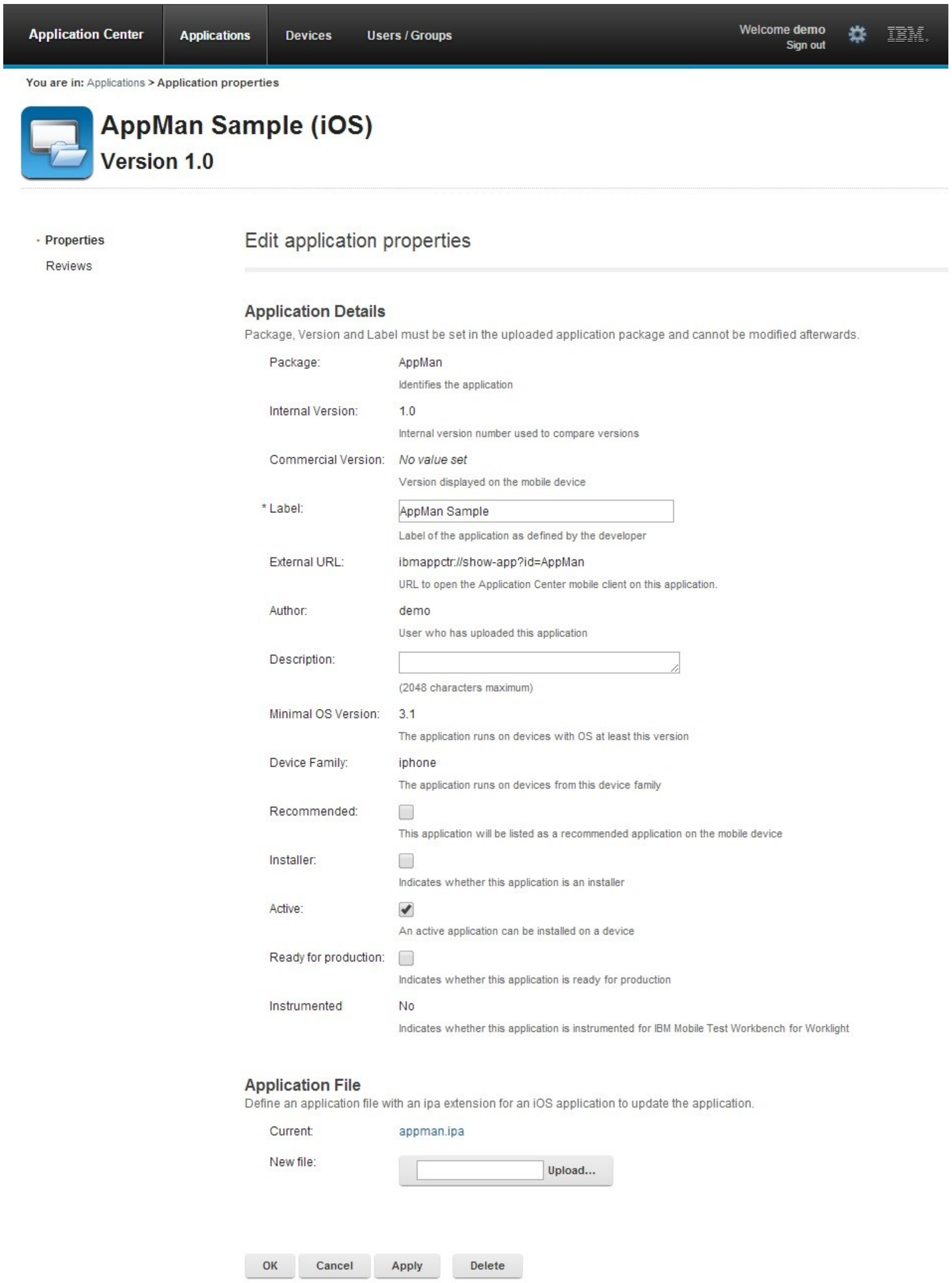
You can edit the properties of an application in the list of uploaded applications.

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 13-27 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.



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:	Upload...

Figure 13-7. 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.

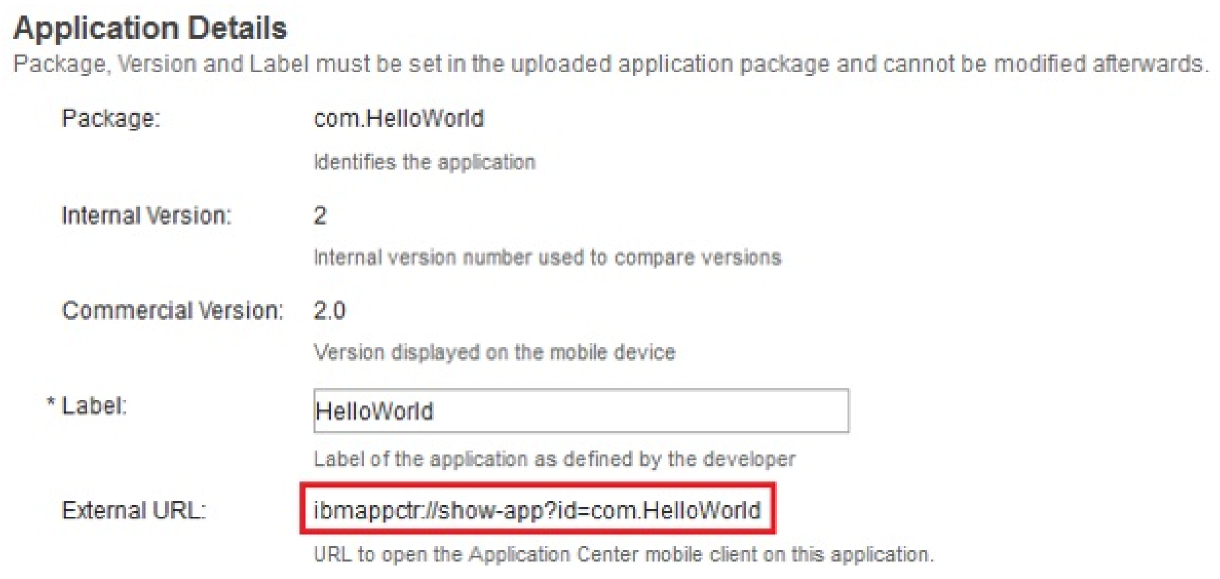
About this task

You can use this procedure to update Android, iOS, and Windows Phone 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 13-22.
2. From the Application Details page, copy the setting of the external URL.



Application Details

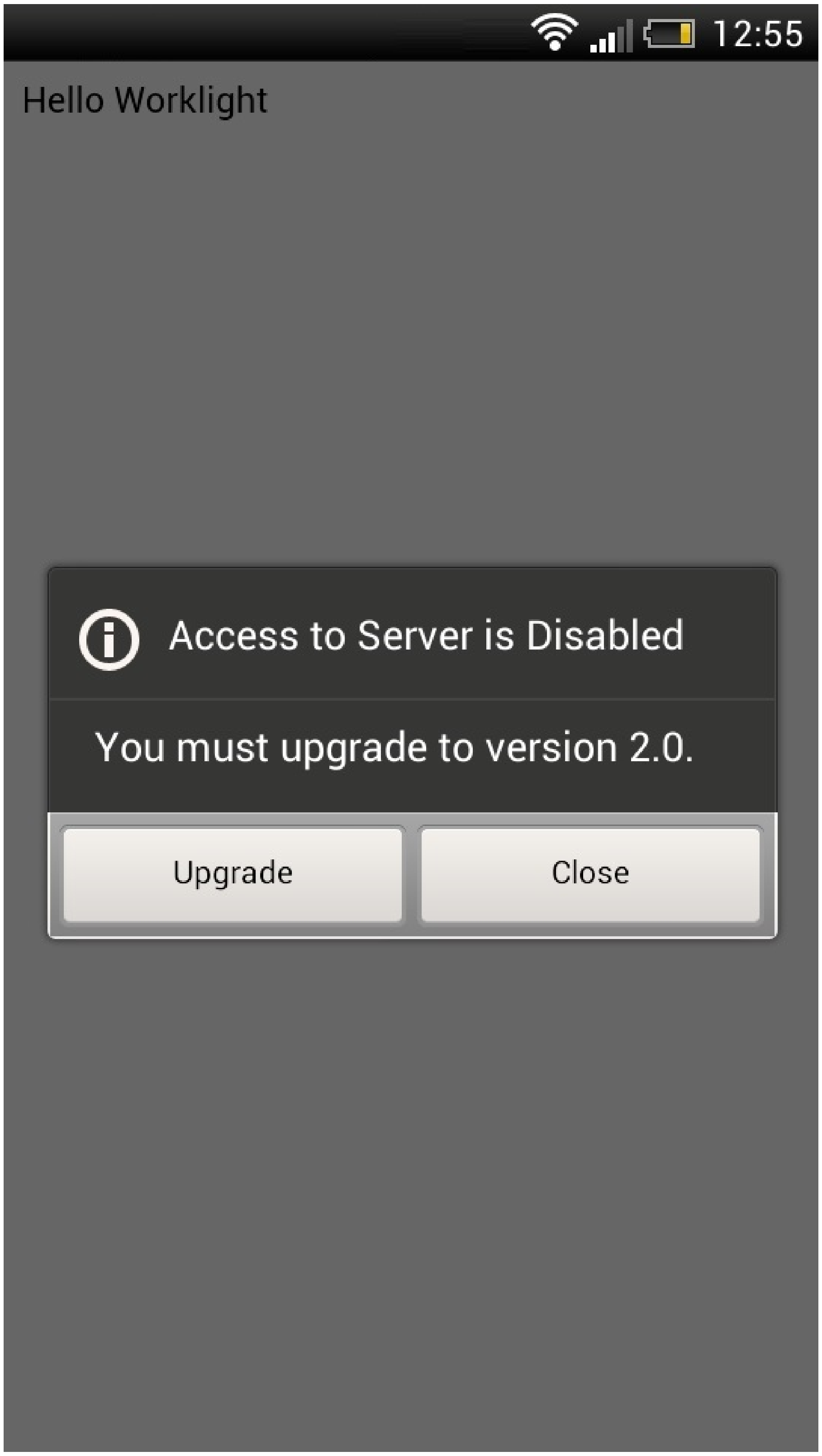
Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

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 13-8. 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.

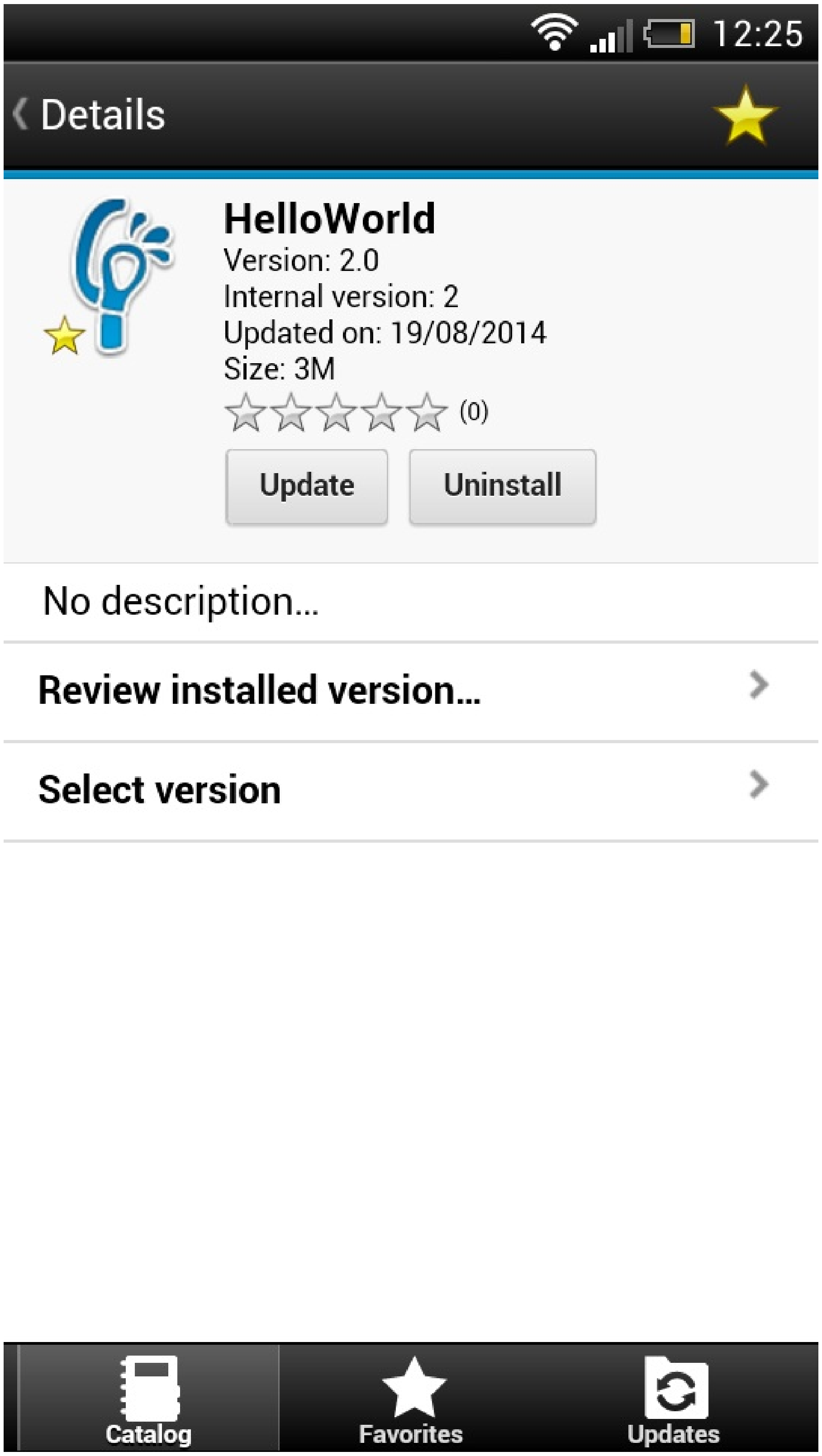


Access to Server is Disabled

You must upgrade to version 2.0.

Figure 13-9. 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
(0)

No description...

Review installed version...

Select version

Figure 13-10. 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.

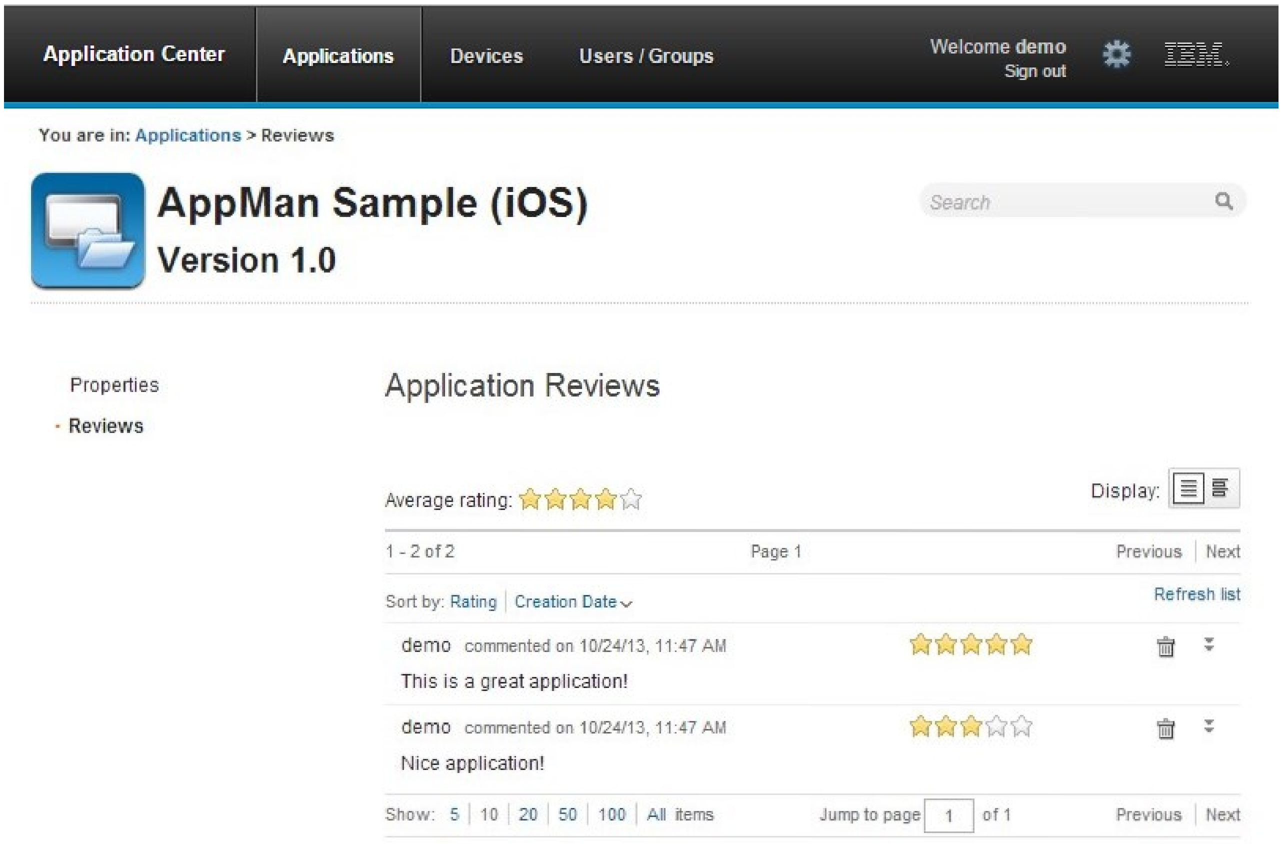
About this task

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 13-11. 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 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 icon to the right of the review that you want to delete.

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.

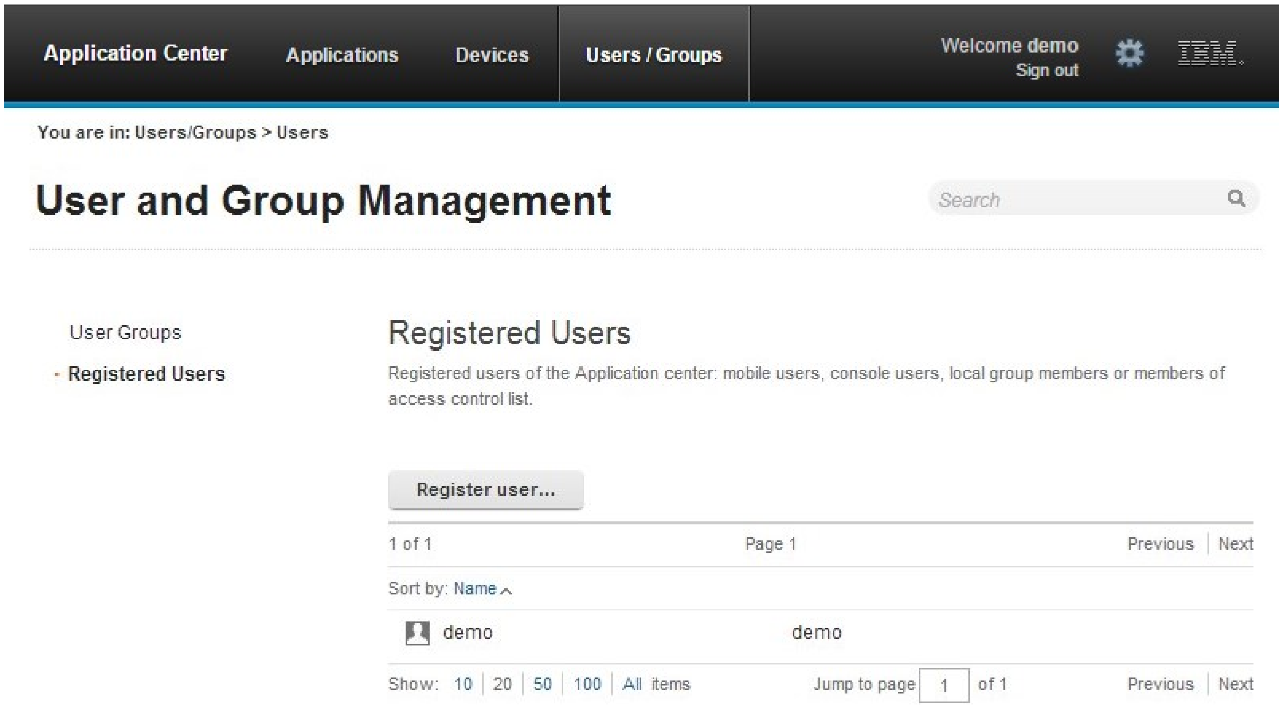
Purpose

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

Registered users of the Application center: mobile users, console users, local group members or members of access control list.

demo demo

Figure 13-12. 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.

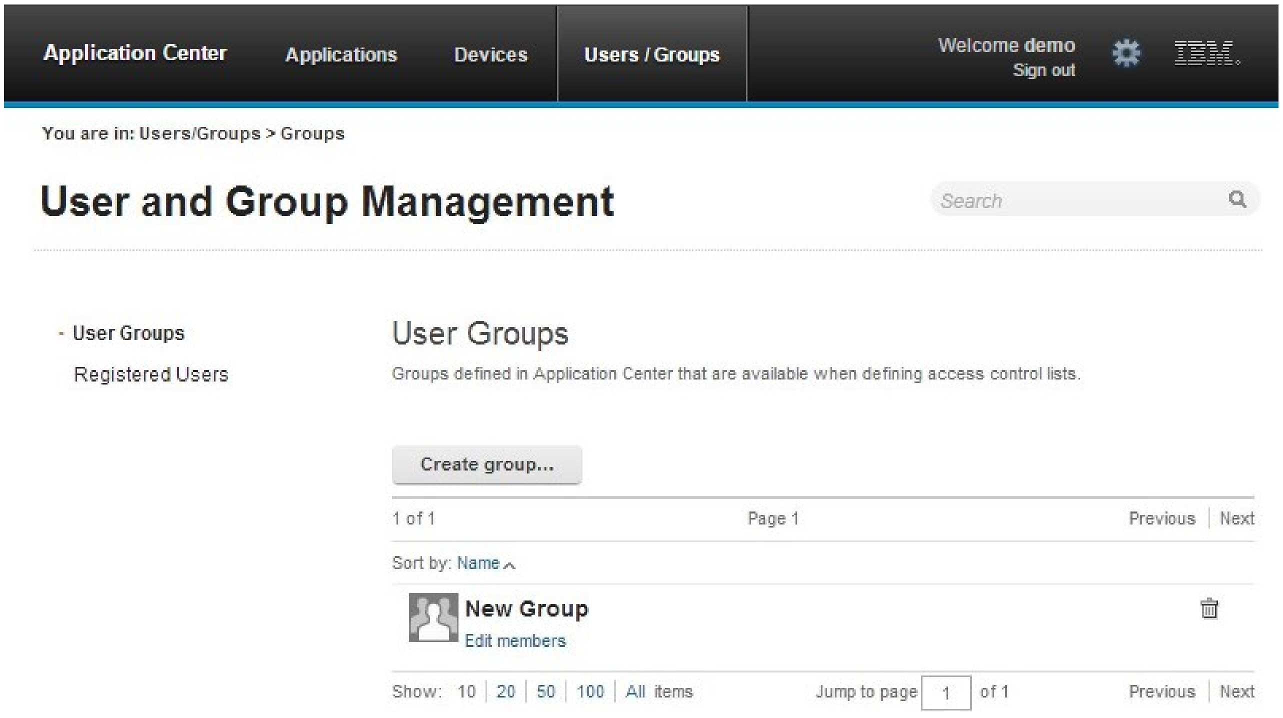
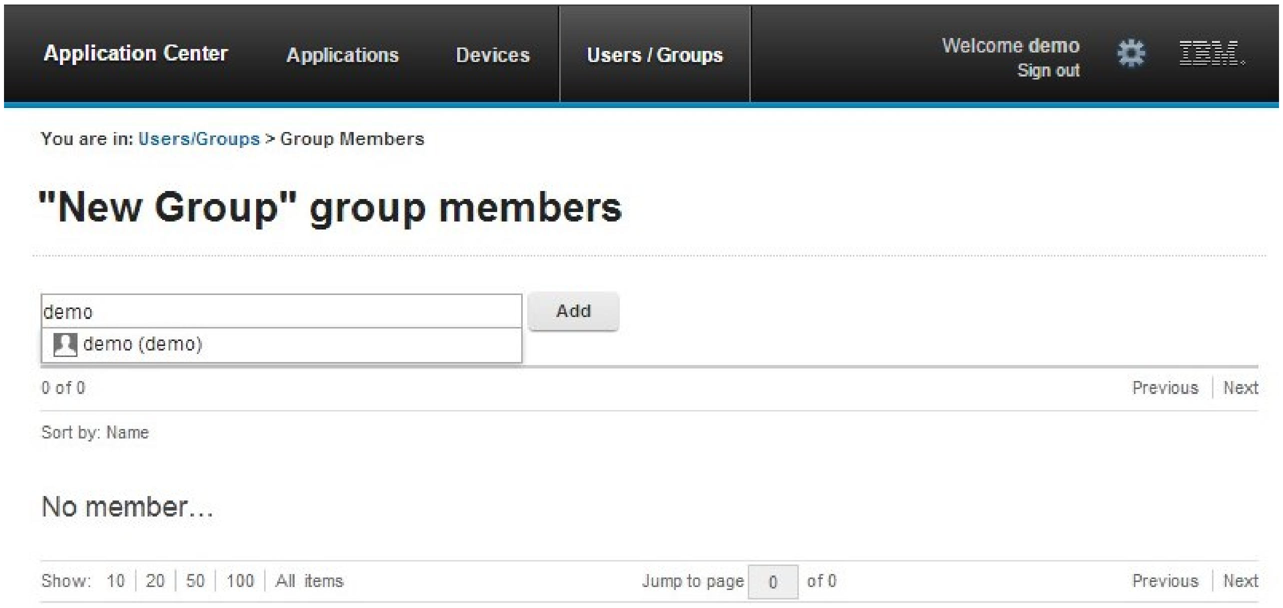


Figure 13-13. Local user groups

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.



"New Group" group members

No member...

Figure 13-14. 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 icon to the right of the user name.

Access control

You can decide whether installation of an application on mobile devices is open to any users or whether you want to restrict the ability to install an application.

Installation of applications on a mobile device can be limited to specific users or available to any users.

Access control is defined at the application level and not at the version level.

By default, after an application is uploaded, any user has the right to install the application on a mobile device.

The current access control for an application is displayed in Available Applications for each application. The unrestricted or restricted access status for installation is shown as a link to the page for editing access control.

Installation rights are only about the installation of the application on the mobile device. If access control is not enabled, everybody has access to the application.

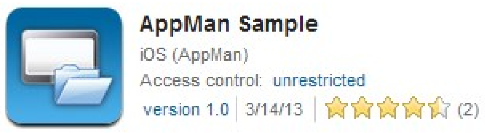
Managing access control

You can add or remove access for users or groups to install an application on mobile devices.

Procedure

You can edit access control:

1. In Application Management under Available Applications, click the **unrestricted** or **restricted** state of Installation of an application.



AppMan Sample
iOS (AppMan)
Access control: unrestricted
version 1.0 | 3/14/13 | (2)

2. Select **Access control enabled** to enable access control.
3. Add users or groups to the access list.

   To add a single user or group, enter a name, select the entry in the matching entries found, and click **Add**.

   If the Application Center is connected to an LDAP repository, you can search for users and groups in the repository as well as locally defined groups. If the repository is not LDAP, you can search only local groups and registered users. Local groups are exclusively defined in the **Users/Groups** tab. When you use the Liberty profile federated registry, you can only search for users by using the login name; the result is limited to a maximum of 15 users and 15 groups (instead of 50 users and 50 groups).

   To register a user at the same time as you add the user to the access list, enter the name and click **Add**. Then you must specify the login name and the display name of the user.

   To add all the users of an application, click **Add users from application** and select the appropriate application.

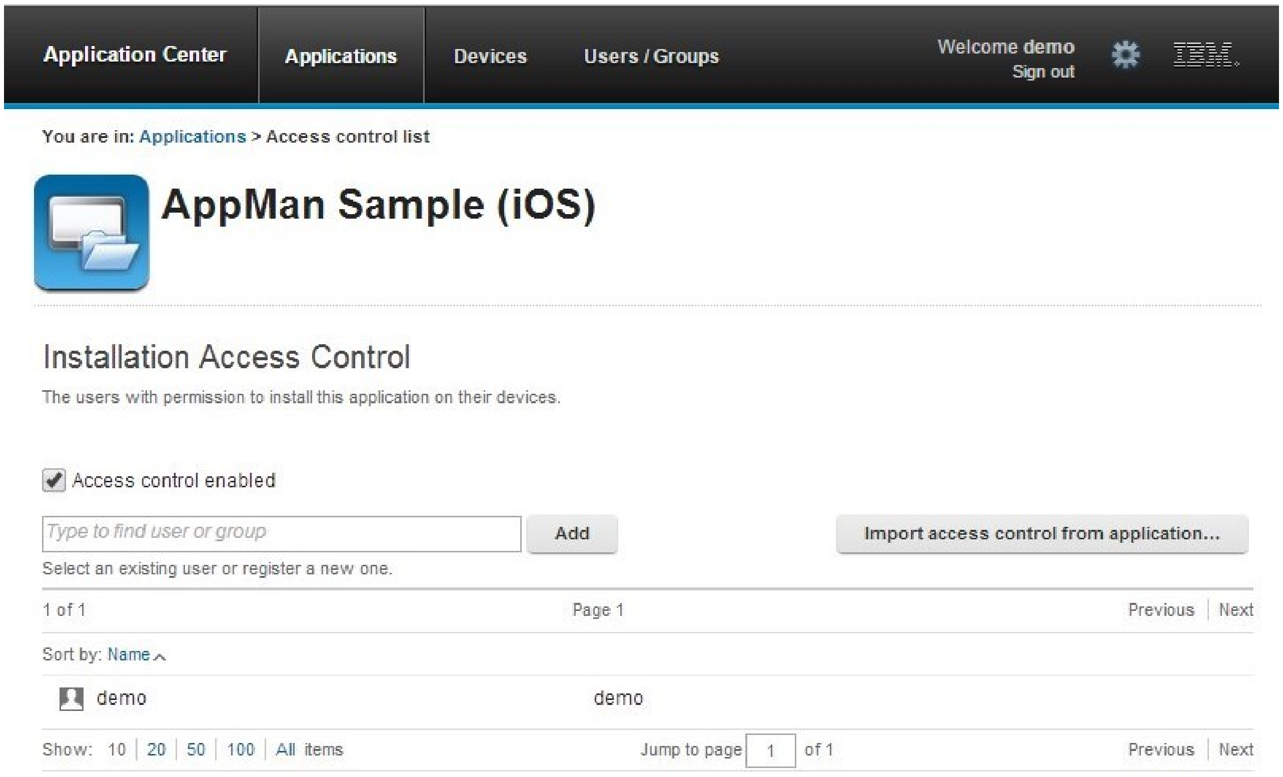

Application Center
Applications
Devices
Users / Groups
Welcome demo
Sign out
You are in: Applications > Access control list
AppMan Sample (iOS)
Installation Access Control
The users with permission to install this application on their devices.
Access control enabled
Type to find user or group
Add
Import access control from application...
Select an existing user or register a new one.
1 of 1
Page 1
Previous
Next
Sort by: Name
demo
demo
Show: 10 | 20 | 50 | 100 | All items
Jump to page 1 of 1
Previous
Next

Figure 13-15. Adding users to the access list

To remove access from a user or group, click the cross icon on the right of the name.

Device Management

You can see the devices that connected to the Application Center from the Application Center mobile client and their properties.

Device Management shows under the **Registered Devices** the list of devices that have connected to the Application Center at least once from the Application Center mobile client.

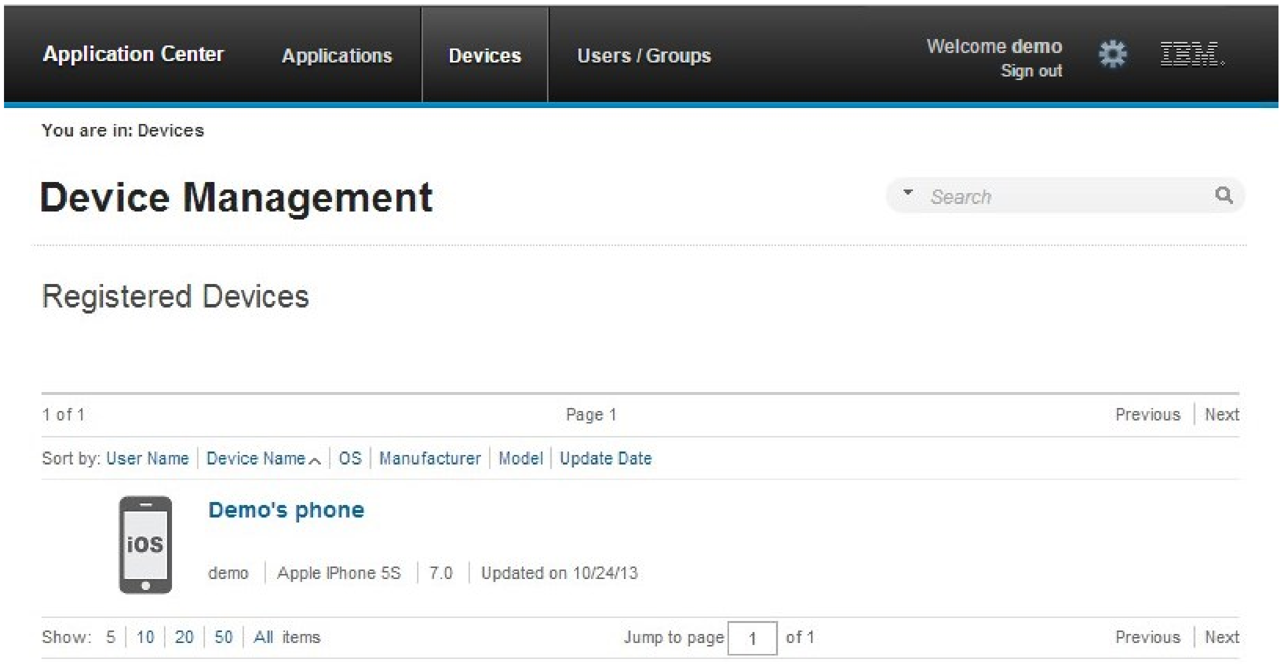
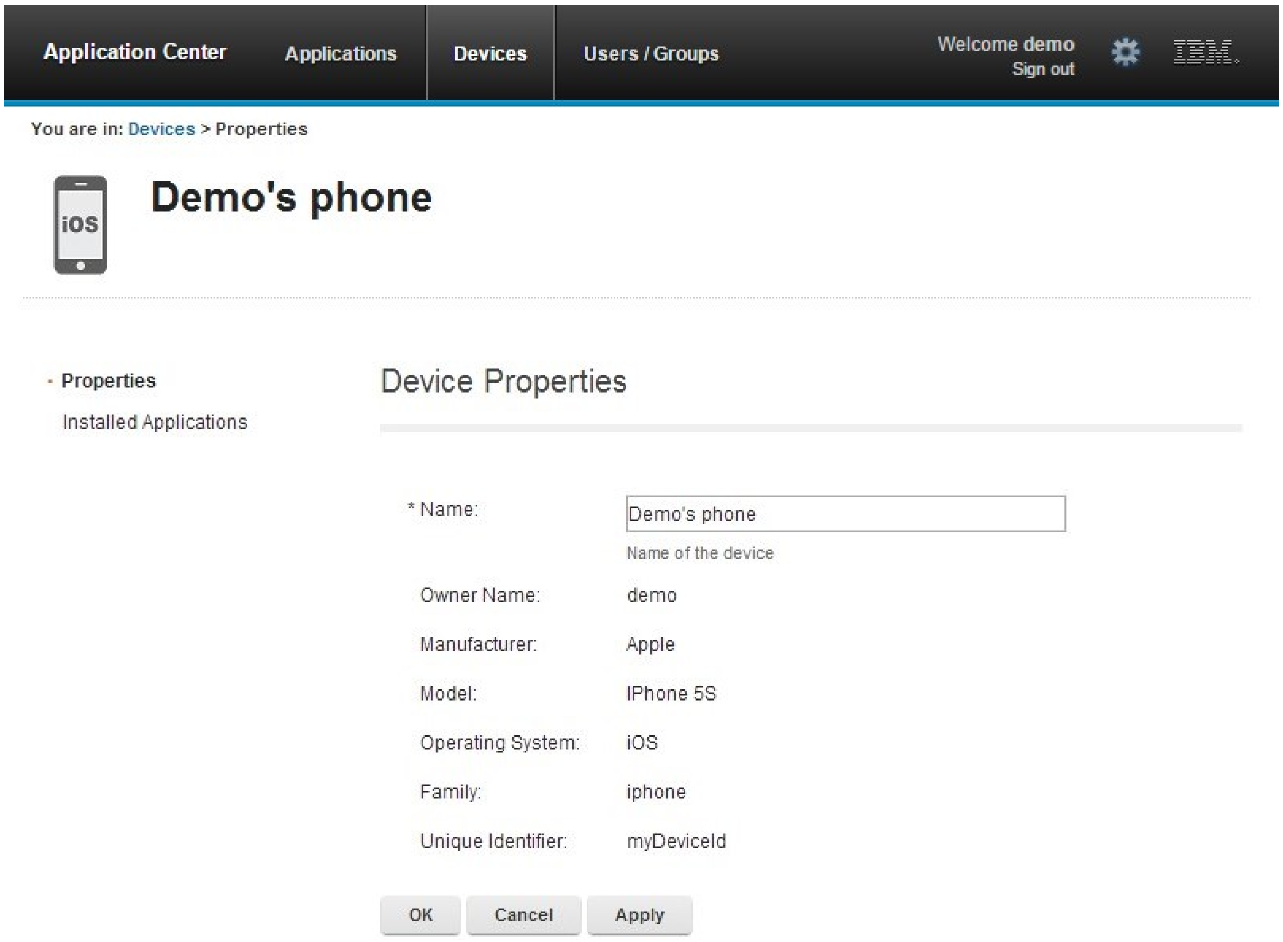


Figure 13-16. The device list

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 13-17. 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.

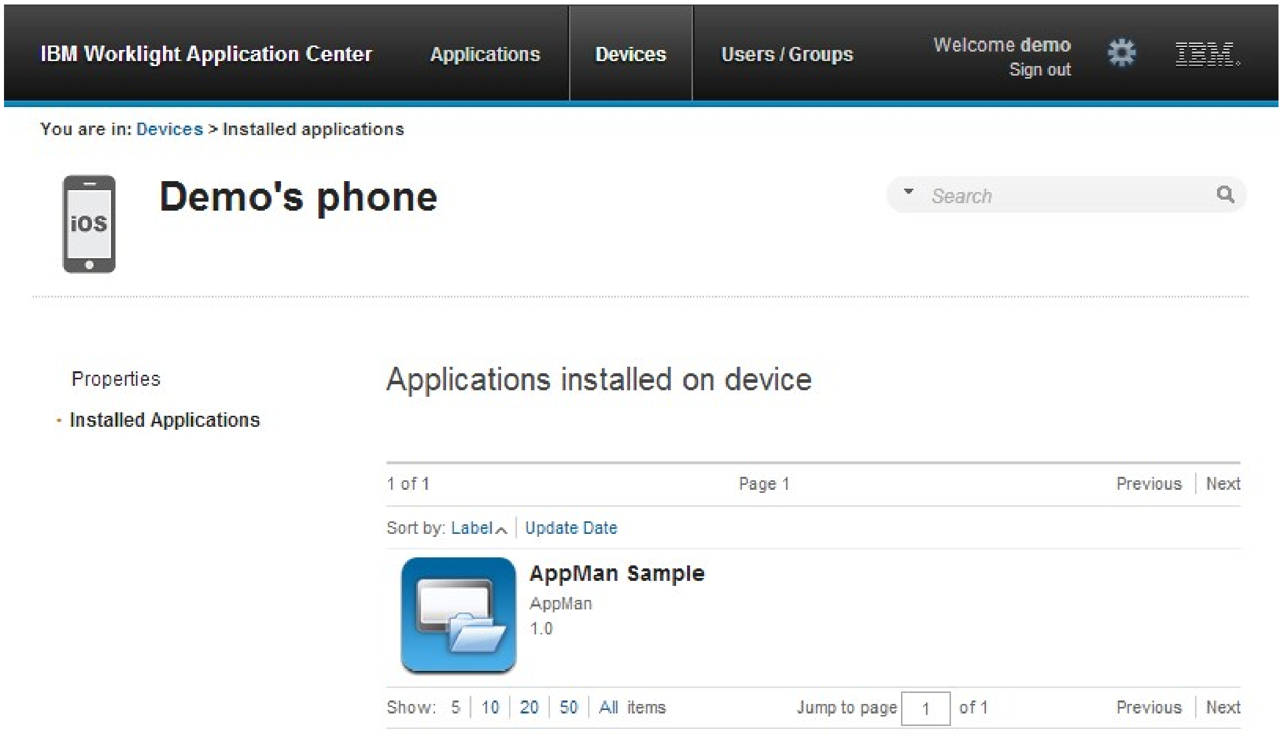


Figure 13-18. Applications installed on a device

Application enrollment tokens in Windows 8 Universal

The Windows 8 Universal operating system requires users to enroll each device with the company before users can install company applications on their devices. One way to enroll devices is by using an application enrollment token.

Purpose

Application enrollment tokens enable you to install company applications on a Windows 8 Universal device. You must first install the enrollment token for a specified company on the device to enroll the device with the company. Then, you can install applications that are created and signed by the corresponding company.

The Application Center simplifies the delivery of the enrollment token. In your role of administrator of the Application Center catalog, you can manage the enrollment tokens from the Application Center console. Once the enrollment tokens are declared in the Application Center console, they are available for Application Center users to enroll their devices.

The enrollment tokens interface available from the Application Center console in the Settings view enables you to manage application enrollment tokens for Windows 8 Universal by registering, updating, or deleting them.

Managing application enrollment tokens

In your role of administrator of the Application Center, you can access the list of registered tokens by clicking the gear icon in the screen header to display Application Center Settings. Then, select **Enrollment Tokens** to display the list of registered tokens.

To enroll a device, the device user must upload and install the token file **before** installing the Application Center mobile client. The mobile client is also a company application. Therefore, the device must be enrolled before the mobile client can be installed.

The registered tokens are available through the bootstrap page at `http://`*`hostname`*`:`*`portnumber`*`/applicationcenter/installers.html`, where *hostname* is the host name of the server hosting the Application Center and *portnumber* is the corresponding port number.

To register a token in the Application Center console, click **Upload Token** and select a token file. The token file extension is `aetx`.

To update the certificate subject of a token, select the token name in the list, change the value, and click **OK**.

To delete a token, click the trash can icon on the right side of the token in the list.

Signing out of the Application Center console

For security purposes, you must sign out of the console when you have finished your administrative tasks.

Purpose

To log out of the secure sign-on to the Application Center console.

To sign out of the Application Center console, click **Sign out** next to the Welcome message that is displayed in the banner of every page.

Command-line tool for uploading or deleting an application

To deploy applications to the Application Center through a build process, use the command-line tool.

You can upload an application to the Application Center by using the web interface of the Application Center console. You can also upload a new application by using a command-line tool.

This is particularly useful when you want to incorporate the deployment of an application to the Application Center into a build process. This tool is located at:

installDir/ApplicationCenter/tools/applicationcenterdeploytool.jar

The tool can be used for application files with extension APK or IPA. It can be used stand alone or as an ant task.

The tools directory contains all the files required to support the use of the tool.
- applicationcenterdeploytool.jar: the upload tool.
- json4j.jar: the library for the JSON format required by the upload tool.
- build.xml: a sample ant script that you can use to upload a single file or a sequence of files to the Application Center.
- acdeploytool.sh and acdeploytool.bat: Simple scripts to call java with applicationcenterdeploytool.jar.

Using the stand-alone tool to upload an application

To upload an application, call the stand-alone tool from the command line.

Procedure

Use the stand-alone tool by following these steps.
1. Add applicationcenterdeploytool.jar and json4j.jar to the java classpath environment variable.
2. Call the upload tool from the command line:
   ```
   java com.ibm.appcenter.Upload [options] [files]
   ```
   You can pass any of the available options in the command line.

Option	Content indicated by	Description
-s	**serverpath**	The path to the Application Center server.
-c	**context**	The context of the Application Center web application.
-u	**user**	The user credentials to access the Application Center.
-p	**password**	The password of the user.
-d	**description**	The description of the application to be uploaded.
-l	**label**	The fallback label. Normally the label is taken from the application descriptor stored in the file to be uploaded. If the application descriptor does not contain a label, the fallback label is used.
-isActive	true or false	The application is stored in the Application Center as an active or inactive application.
-isInstaller	true or false	The application is stored in the Application Center with the "installer" flag set appropriately.

Option	Content indicated by	Description
-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 Android application package (.apk) files or iOS application (.ipa) files.

In this example user `demo` has the password `demopassword`. Use this command line.

```
java com.ibm.appcenter.Upload -s http://localhost:9080 -c applicationcenter -u demo -p 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.
-u	**user**	The user credentials to access the Application Center.
-p	**password**	The password of the user.

Option	Content indicated by	Description
-y		Disable SSL security checking, which allows publishing on secured hosts without verification of the SSL certificate. Use of this flag is a security risk, but may be suitable for testing localhost with temporary self-signed SSL certificates.

You can specify files or the application package, operating system, and version. If files are specified, the package, operating system and version are determined from the file and the corresponding application is deleted from the Application Center. If applications are specified, they must have one of the following formats:

`package@os@version`: This exact version is deleted from the Application Center. The version part must specify the “internal version”, not the “commercial version” of the application.

`package@os`: All versions of this application are deleted from the Application Center.

`package`: All versions of all operating systems of this application are deleted from the Application Center.

Example

In this example, user **`demo`** has the password **`demopassword`**. Use this command line to delete the 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 2-7.

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.
context	The context of the Application Center. The default value is /applicationcenter.
loginUser	The user name with permissions to upload an application.

Parameters of Ant task	Description
loginPass	The password of the user with permissions to upload an application.
forceOverwrite	If this parameter is 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 `.apk` or `.ipa` file to be uploaded to the Application Center or to be deleted from the Application Center. This parameter has no default value.
fileset	To upload or delete multiple files.
application	The package name of the application; this parameter is available only in the **delete** Ant task.
os	The operating system of the application. (For example, Android or iOS.) 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 Android, iOS, Windows Phone, or Windows device. Only Windows Phone 8 is supported by the current version of the Application Center. You use the mobile client to list the catalog of available applications in the Application Center. You can install these applications on your device. The mobile client is sometimes referred to as the Application Center installer. This application must be present on your device if you want to install on your device applications from your private application repository.

Prerequisites

Your system administrator must give you a user name and password before you can download and install the mobile client. The user name and password are required whenever you start the mobile client on your device. For Windows Store applications, the user name and password are required for the mobile client only at run time. For security reasons, do not disseminate these credentials. These credentials are the same credentials used to log in to the Application Center console.

Installing the client on an Android mobile device

You can install the mobile client, or any signed application marked with the installer flag, on your Android mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

Procedure

1. Start the browser on your mobile device.
2. Enter the following access URL in the address text field: `http://`*`hostname`*`:`*`portnumber`*`/applicationcenter/installers.html`

   Where *hostname* is the address of the server and *portnumber* is the number of the port where the Application Center is installed. Your system administrator can provide this information.

   The Application Center also provides an alternative URL for installing the client on a mobile device: `http://hostname:portnumber/applicationcenter/inst.html`. The page of this URL works better with some older or some nonstandard mobile web browsers. If the page `installers.html` does not work on your mobile device, you can use `inst.html`. This page is provided in English only and is not translated into other languages.

   If you try to open the page with HTTPS and use self-signed certificates, older Android browsers cannot open the page. In this case, you must use a non self-signed certificate or use another browser on the Android device, such as Firefox, Chrome, or Opera. In Android 4 and later, the Android 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 13-51.

   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.
4. If the web server uses a self-signed CA certificate, install the certificate at least once on the device.

   The Application Center administrator should provide the certificate; see “Managing and installing self-signed CA certificates in an Application Center test environment” on page 6-260 for details.
   a. Tap the **SSL-Certificate** tab and select the certificate.
   b. Tap **Install**. You must only perform this action once for the device. You can verify whether the certificate is installed by looking in **Settings** > **Security** > **Trusted Credentials** > **User** on the device. This view shows the SSL certificates that the user has installed on the device. If the self-signed CA certificate is not installed on the device, the Android 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 13-27.

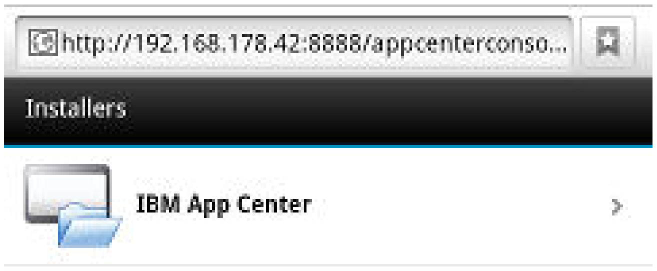
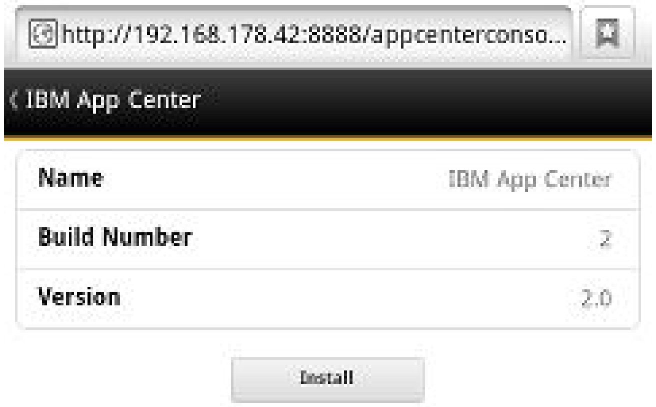


Figure 13-19. List of available mobile client applications to install

5. Select an item in the list to display the application details.

   Typically, these details include the application name and its version number.


http://192.168.178.42:8888/appcenterconso...

IBM App Center

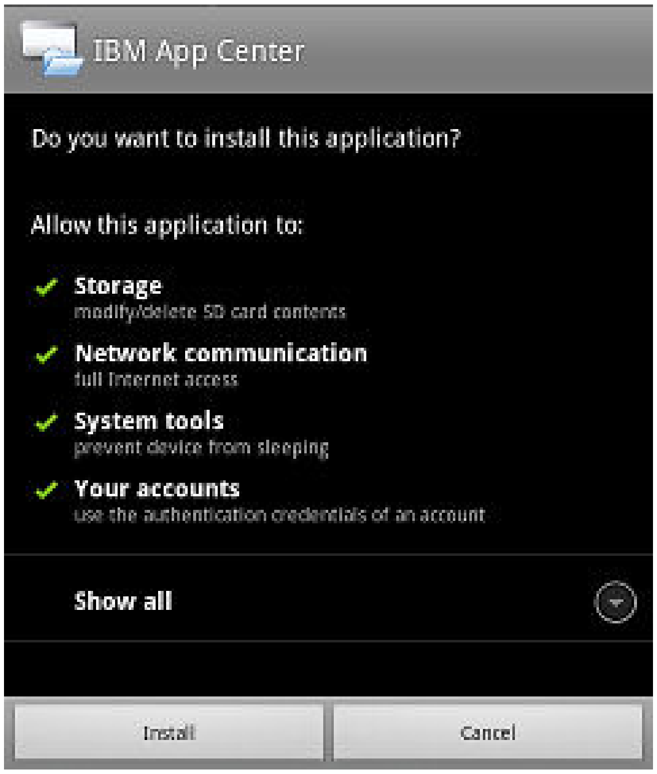
Name	IBM App Center
Build Number	2
Version	2.0

Figure 13-20. Application details

6. Tap **Install Now** to download the mobile client.

   On newer Android devices, a question might request permission for Chrome to access media files on the device. Select **YES**. A warning about potential harmful files might be displayed. Select the option to keep the APK file anyway.
7. Launch the **Android Download** applications.
8. Select the Application Center client installer.

   You can see the access granted to the application when you choose to install it.


IBM App Center

Do you want to install this application?

Allow this application to:

✓ **Storage**
modify/delete SD card contents

✓ **Network communication**
full Internet access

✓ **System tools**
prevent device from sleeping

✓ **Your accounts**
use the authentication credentials of an account

Show all

Figure 13-21. Installation of the mobile client on Android

9. Select **Install** to install the mobile client.
10. When the application is installed, select **Open** to open the mobile client or **Done** to close the Downloads application.

Results

The APK file might fail for one of the following reasons:
- The device does not have enough free memory.
- The SSL certificate of the server is not known to the device.

The first time that you install an app through the Downloads application, you might receive a request to confirm whether Google should regularly check the device activity for security problems. You can accept or decline according to your preference. The Application Center client is unaffected by your choice.

The installation might be blocked for one of the following reasons:
- The device does not permit installation from unknown sources. Go to **Settings** > **Security** on the device and enable **Unknown sources (Allow installation from unknown sources)**.
- The device has the same app already installed, but it was signed by a different certificate. In this case, you must remove the app before you install it on the device with another signed certificate.

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 6-257.

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 6-252.

Procedure

Installing the mobile client on an iOS device is similar to installing it on Android, but with some differences. 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`. The 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 you can proceed to the website after confirmation that you consent to an unsafe connection.
3. Enter your user name and password.

   See the prerequisites in "The mobile client" on page 13-51.

   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.
- 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 6-260 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 13-27.
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**.

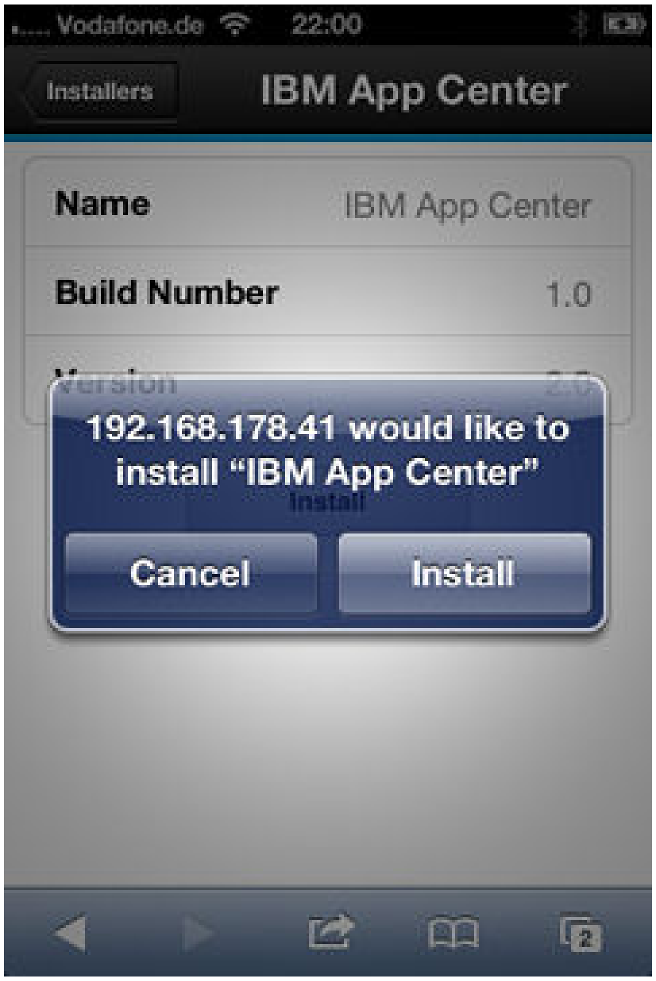


Figure 13-22. Confirm app to be installed

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 (Android, iOS, Windows Phone)" on page 13-8.

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 the client on Windows 8 Universal

You can install the mobile client, or any signed application marked with the installer flag, on Windows 8 Universal by entering the access URL in your browser, entering your credentials, and completing the required steps. The company account must be preinstalled on your mobile device.

Before you begin

Before you can install apps published by your company, you must add the company account to your mobile device. You must download an application enrollment token (AET) to your Windows Phone device. This AET must already be present on the IBM MobileFirst Platform Server. It is uploaded to the MobileFirst Server by using the Application Center console. See "Application enrollment tokens in Windows 8 Universal" on page 13-44 for details.

Procedure

1. Start the browser on your mobile device.
2. Enter the following access URL in the address text field: `http://`*`hostname`*`:`*`portnumber`*`/applicationcenter/installers.html`.

   Where *hostname* is the address of the server and *portnumber* is the number of the port where the Application Center is installed. Your system administrator can provide this information.

   The Application Center also provides an alternative URL for installing the client on a mobile device: http://hostname:portnumber/applicationcenter/inst.html. The page of this URL works better with some older or some nonstandard mobile web browsers. If the page `installers.html` does not work on your mobile device, you can use `inst.html`. This page is provided in English only and is not translated into other languages.
3. Enter your credentials to authorize access to the server.

   On the lower part of the screen, a toolbar contains an **Installers** tab and a **Tokens** tab.

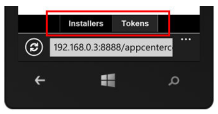
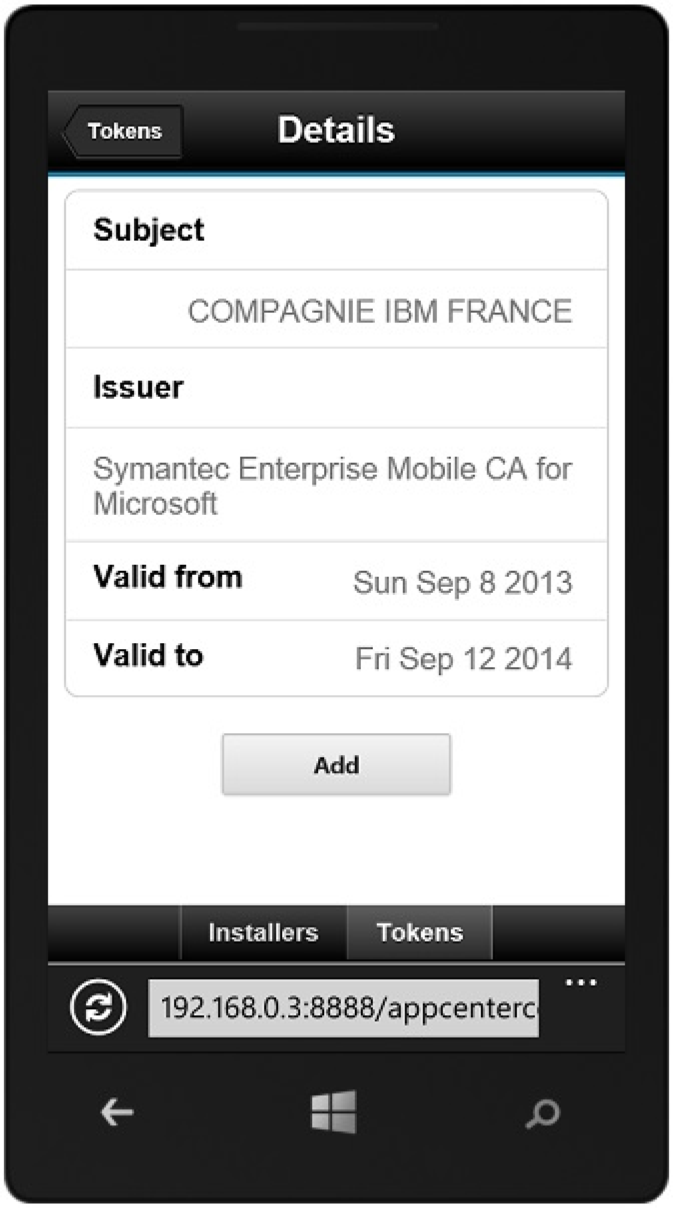


Figure 13-23. Preparing to install tokens and applications on a Windows Phone device

4. Tap **Tokens** and select an application enrollment token in the list of available tokens to display the token details.



Subject

COMPAGNIE IBM FRANCE

Issuer

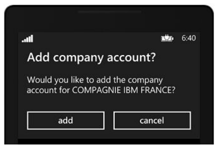
Symantec Enterprise Mobile CA for Microsoft

Valid from Sun Sep 8 2013

Valid to Fri Sep 12 2014

Figure 13-24. AET details on a Windows Phone device

5. Tap **Add** to download the application enrollment token.
6. Tap **Add** to add the company account.



Add company account?

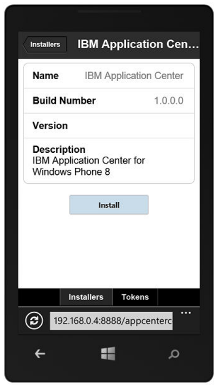
Would you like to add the company account for COMPAGNIE IBM FRANCE?

Figure 13-25. Adding a company account in Windows 8 Universal device

Windows Phone 8 does not provide any feedback about adding the company account.

7. Tap the Back icon to return to the details of application enrollment tokens.

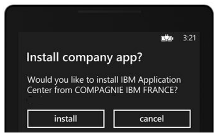
8. Tap **Installers** and select the mobile client application in the list of available applications. The application details are displayed.
9. Tap **Install** to download the selected application.



Name	IBM Application Center
Build Number	1.0.0.0
Version	
Description IBM Application Center for Windows Phone 8	

Figure 13-26. The application selected to download on a Windows Phone device

10. Tap **Install** to install the application.



Install company app?

Would you like to install IBM Application Center from COMPAGNIE IBM FRANCE?

Figure 13-27. Installing the downloaded application on a Windows Phone device

Windows 8 Universal does not provide any feedback about installing the application.

Tip: When you install a company application on a device, the device must connect to the Microsoft server to verify the application signature. Sometimes, the Microsoft server does not respond. In this case, you can try the installation again a few minutes later. See "Installing an application on a Windows Phone device" on page 13-70 for the possible error messages.

Results

When the installation is finished, the mobile client application should be available in your applications list in Windows Phone.

The Login view

In the Login view, you can access the fields that are required to connect to the server to view the list of applications available for your device.

Use the **Login** view to enter your credentials to connect to the Application Center server to view the list of applications that are available for your device.

The **Login** view presents all the mandatory fields for the information that is 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 later start the application, the login page is not displayed and the previous credentials are used. If login fails, the login view is displayed.

User name and password
Enter your credentials for access to the server. They are the same user name and password as the ones that were granted by your system administrator for downloading and installing the mobile client.

Application Center server address
The Application Center server address is composed of the following elements:

- 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 13-6 for information on filling parts of the address in advance, or hardcode the address and hide the associated fields.

Secure Socket Layer (SSL)
SSL is mandatory on iOS devices. Therefore, this option is not displayed in the login view.

On the other supported operating systems, select SSL to turn on the SSL protocol for communications over the network. If you tap this field again when SSL is selected, SSL switches off.

SSL selection is available for cases where the Application Center server is configured to run over an SSL connection. Selecting SSL when the server is not configured to handle an SSL layer prevents you from connecting to the server. Your system administrator can inform you whether the Application Center runs over an SSL connection.

Connecting to the server

To connect to the server:

1. Enter your user name and password.
2. Enter your Application Center server address.
3. If your configuration of the Application Center runs over the SSL protocol, select **SSL**.
4. Tap **Log in** to connect to the server.

If this login is successful, the user name and server address are saved to fill the fields when you start the client afterwards.

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.

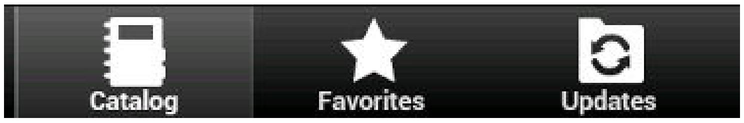


Figure 13-28. Views in the client application (Android, iOS, and Windows Phone operating systems)

Use these views to communicate with a server to send or retrieve information about applications or to manage the applications that are located on your device.

The Windows 8 client home screen displays up to six applications in each category. On the Windows 8 client, if you want the full list of applications in a category, click the title of the category.

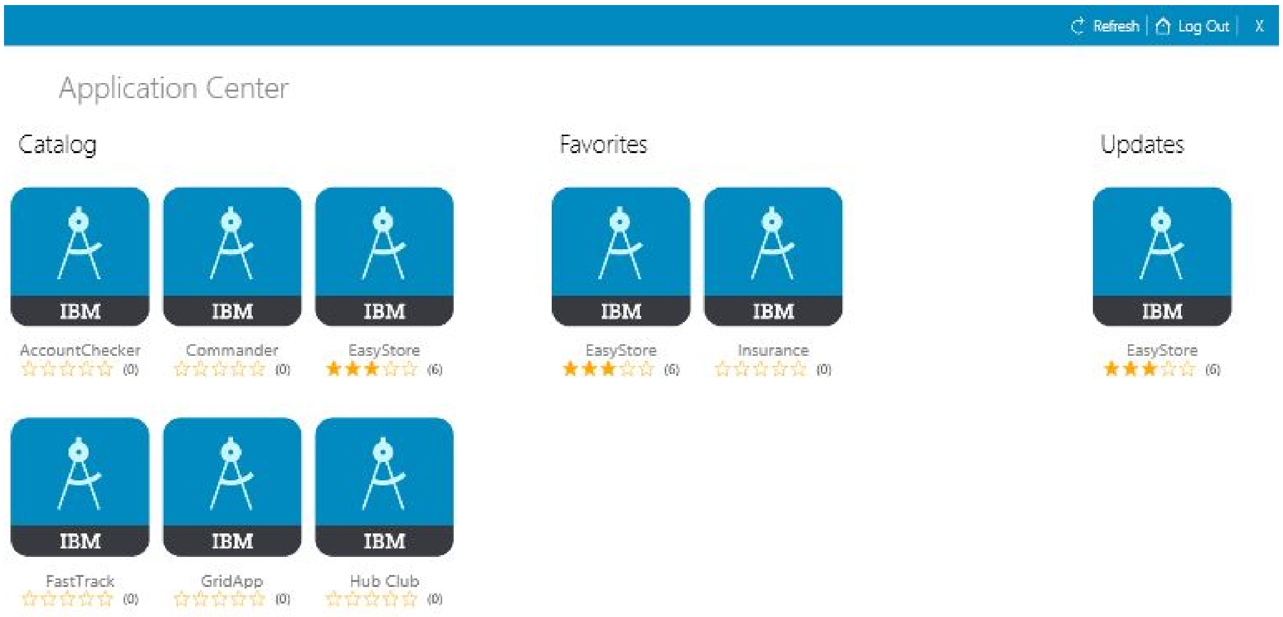


Figure 13-29. Client home screen on Windows 8

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

The layout of the views is specific to the Android, iOS, Windows Phone, or Windows 8 environment, even though the common functions that you can perform in the views are the same for all operating systems. Different device types might have different page displays. On the phone, a list is displayed. On a tablet, a grid of applications is used.

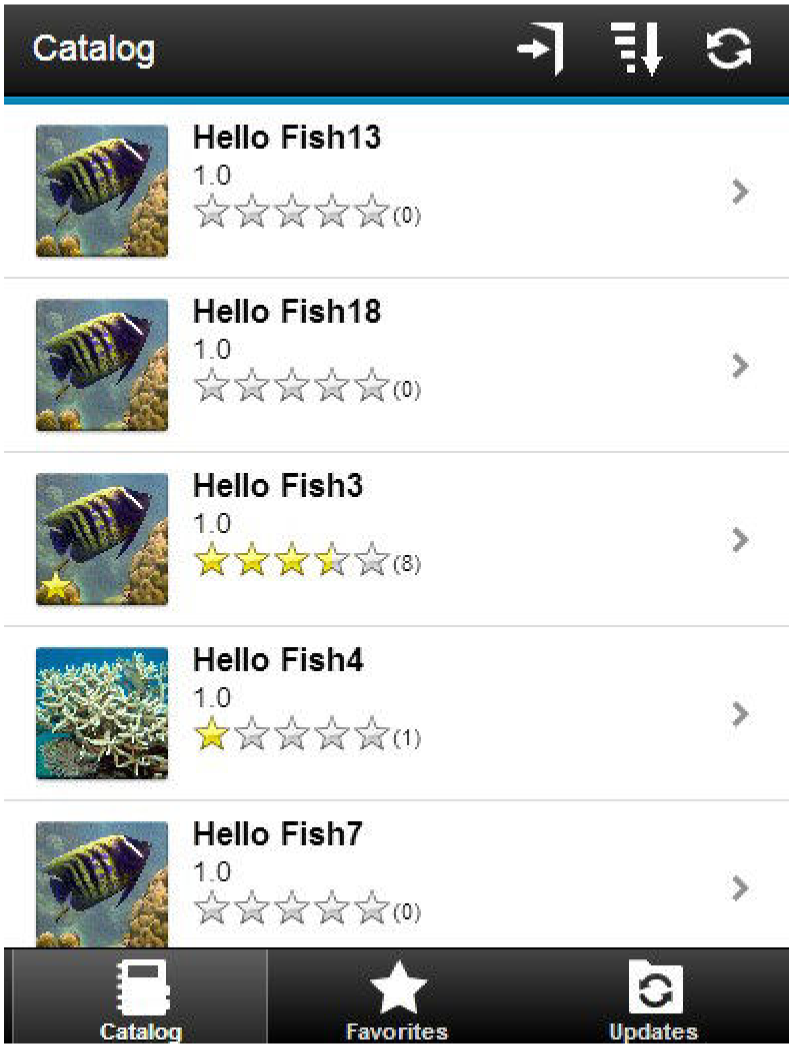


Figure 13-30. Catalog view on a phone

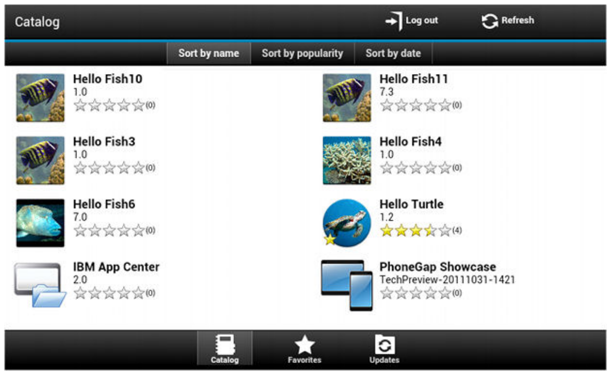


Figure 13-31. Catalog view on a tablet

Features of the views

On an Android or iOS tablet, you can sort the lists by tapping one of the sort criteria.

On a Windows Phone, Android, or iOS phone, sort criteria are available through the sort button.

On the Windows 8 client, you can sort the list of applications within a category. To sort the applications, select from the list of sort criteria in the **Sort By** field.

Sort By: Last Update

Applications that are marked as favorites are indicated by a star that is 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 13-6 for how to show the rating of all versions of the application instead of the latest version only.

Tapping an application in the list opens the Details view of the latest installed version of this application.

To refresh the view, tap the refresh button: or, on Windows 8, Refresh.

To return to the login page:

- In Android, iOS, and Windows Phone applications, tap the logout button.
- In the Windows 8 version of the client, tap the logout button.
  Log Out

The Details view

Tapping an application in the Catalog, Favorites, or Updates view opens the Details view where you can see details of the application properties. Details of the application version are displayed in this view.

On Android, iOS, and Windows Phone clients, the following details of the application version are displayed:
- The name of the application.
- Commercial version: the published version of the application.
- Internal version: on Android, the internal version identification of the application; on iOS, the build number of the application. See "Application properties" on page 13-27 for technical details about this property on all operating systems.
- Update date.
- Approximate size of the application file.
- Rating of the version and number of ratings received.
- Description of the application.

On Windows 8 client the following details of the application version are displayed:
- Application name.
- Version.
- Vendor name.
- Update date.
- Rating of the version and the number of ratings received.
- Existing reviews of either the current version or of all the versions of the current application.

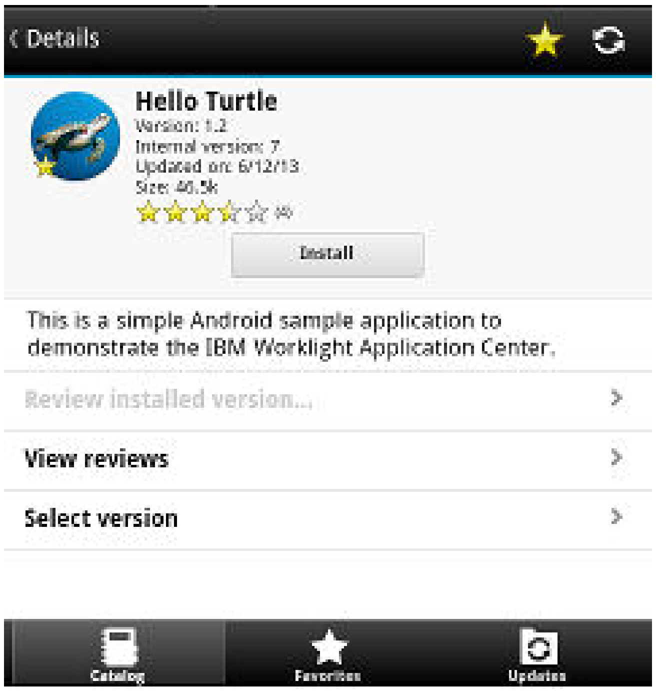
You can take the following actions in this view.
- Install, upgrade, downgrade, or uninstall an application version.
- Cancel the current operation in progress (if available).
- Rate the application version if it is installed on the device.
- List the reviews of this version or of all versions of the application.
- Show details of a previous version.
- Mark or unmark the application as a favorite app.
- Refresh the view with the latest changes from the Application Center server.

Installing an application on an Android device

From the **Details** view, you can install an application on your Android device.

About this task

In the **Details** view, if a previous version of the application is not installed, you can install this application version on your Android device.



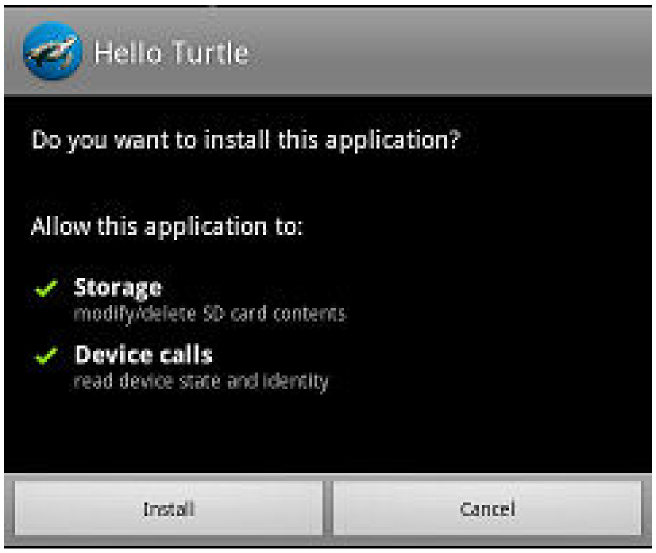
This is a simple Android sample application to demonstrate the IBM Worklight Application Center.

Figure 13-32. Details view of an app version shown on your Android device

Procedure

1. In the **Details** view, tap **Install**.

   The application is downloaded. You can tap **Cancel** in the **Details** view at any time during the download to cancel the download. (The **Cancel** button appears only during the installation steps.) If you let the download complete, you will see the rights that are granted to the application.



Do you want to install this application?

Allow this application to:

- Storage
  modify/delete SD card contents
- Device calls
  read device state and identity

Figure 13-33. Application rights on your Android device

2. Tap **Install** to confirm installation of the application or **Cancel** to cancel installation..

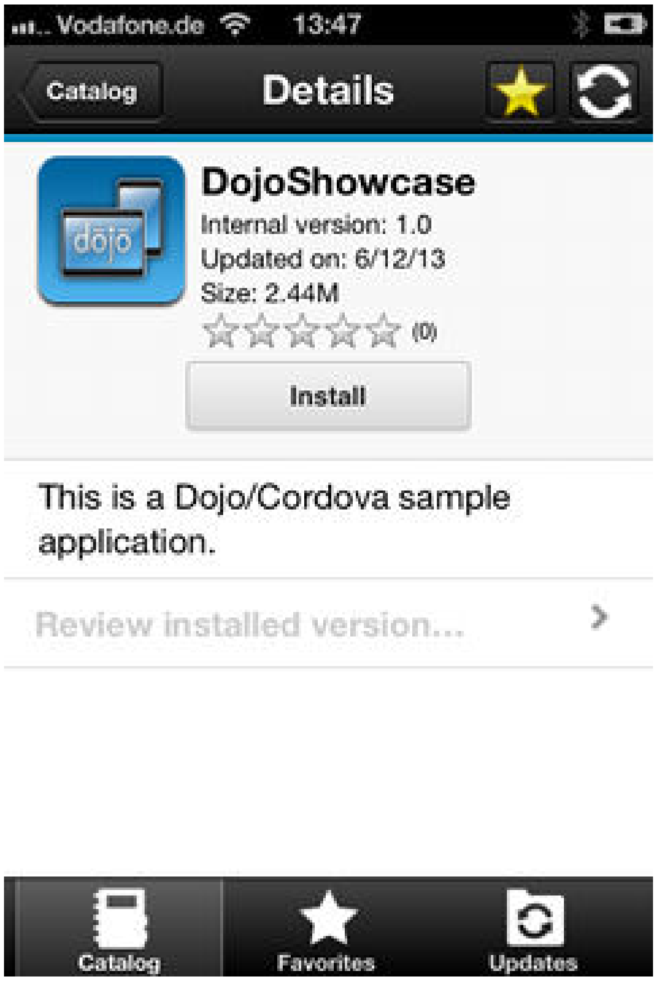
   Depending on the action taken, the application is installed or not. When the application is successfully installed, it is also marked as a favorite app.

If you selected **Cancel**, in the application rights confirmation panel, you can tap **Cancel** in the **Details** view at any time to notify the application that the installation has been canceled. The **Cancel** button appears in the **Details** view only during the installation steps.

Installing an application on an iOS device

From the **Details** view, you can install an application version on your iOS mobile device.

About this task



DojoShowcase
Internal version: 1.0
Updated on: 6/12/13
Size: 2.44M
(0)

This is a Dojo/Cordova sample application.

Figure 13-34. 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 6-257.

Procedure

1. In the **Details** view, tap **Install**. You are requested to confirm the download and installation of the application version.
2. Tap **Install** to confirm download and installation of the application version or **Cancel** to cancel the installation.

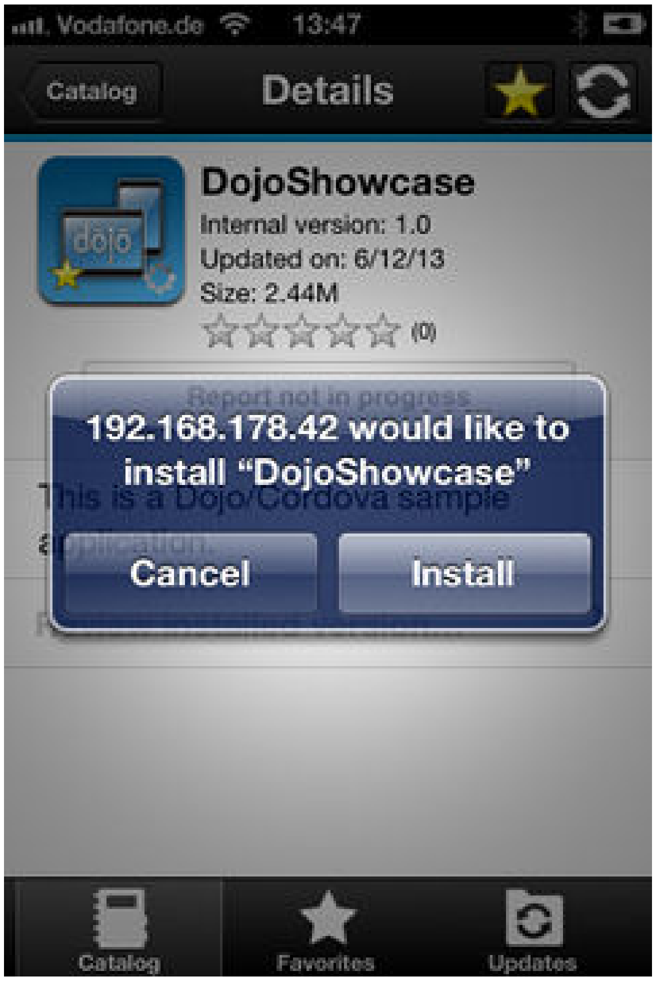


Figure 13-35. Canceling application installation on your iOS device

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, no **Uninstall** button is available. 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 (Android, iOS, Windows Phone)” on page 13-8.

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.

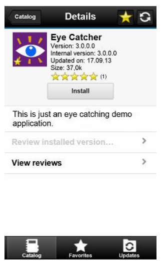
After the trust is confirmed, no application that uses that provisioning profile shows the Untrusted Enterprise Developer message. For more information, see the Apple web site at `https://support.apple.com/en-us/HT204460`.

Installing an application on a Windows Phone device

From the Details view, you can install a company application on your Windows phone device.

About this task

The Details view of the selected application displays information about the application that you want to install.


Eye Catcher
Version: 3.0.0.0
Internal version: 3.0.0.0
Updated on: 17.09.13
Size: 37,0k
(1)

This is just an eye catching demo application.

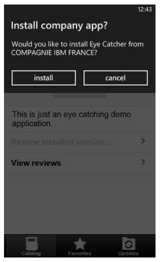
Figure 13-36. Details view of a version of a company application for installation on a Windows Phone device

Procedure

1. In the **Details** view, tap **Install**. The application is downloaded and installed. You can tap **Cancel** at any time during the downloading of the application to cancel the activity. **Cancel** appears only during the downloading step of the installation process.

   At the beginning of the installation process, you are requested to confirm whether you want to add the company application to the applications installed on your mobile device.
2. Tap **Install** to confirm installation of the application or **Cancel** to cancel the installation.

   The application is marked as a favorite app.


12:43

Install company app?

Would you like to install Eye Catcher from COMPAGNIE IBM FRANCE?

This is just an eye catching demo application.

Figure 13-37. Confirming or canceling installation of a company application on a Windows Phone device

Tip: When you install a company application on a device, the device must connect to the Microsoft server to verify the application signature. Sometimes, the Microsoft server does not respond. In this case, you can try the installation again a few minutes later.

The possible error messages are:

- `There's a problem with this company app. Contact your company's support person for help.`

  You are probably using an unsigned Windows Phone application package (.xap) file. You must sign application package (.xap) files before using them in the Application Center. This message might also occur if the Microsoft server does not respond and the signature of the company application cannot be validated. In this case, try the installation again a few minutes later.
- `Before you install this app, you need to add ... company account.`

  The Windows Phone application package (.xap) file is signed, but the device is not enrolled for company applications. You must first install on the device the application enrollment token of the company.
- `We haven't been able to contact the` *`company`* `account to make sure you can install this app. ...`

  Either the company account is expired or blocked, or the Microsoft server is temporarily not responding. Make sure that your device is connected to the internet and connected to the Microsoft server, and try again.

Note: If a device is registered with several company accounts, the Windows Phone operating system might display the wrong company account in the message `Would you like to install` *`application`* `from company` *`name`*?. This message is outside the control of the Application Center. This situation is a display problem only and does not affect the functionality.

Results

Depending on the action that you take, the application is installed or not.

Tip: The install process will not work if the PFX certificate used to code sign the application package (.xap) file of the application that you want to install has expired. Windows Phone operating system returns an error with `HRESULT 0x81030110`. When you renew your PFX certificate, you must code sign again with this new certificate all the deployed applications that you have in your Application Center catalog.

When you renew your PFX code-signing certificate, you must also renew the enrollment token and deploy it on the Application Center console. Devices must also be re-enrolled to the company account with this new token. Users of devices enrolled with an expired token cannot install any applications.

In Windows Phone 8.1, if the Application Center client is not code signed (for example, when you debug it in Visual Studio), you cannot install any application by using this unsigned client. In this case, the Windows Phone operating system returns an error with `HRESULT 0x800703F0`. Before installing applications in Windows Phone 8.1, you must code sign the application package (.xap) file of the client.

Installing a Windows Store application on a Windows device

Use sideloading to install Windows Store apps through Application Center.

Before you begin

You must check that your configuration satisfies the application sideloading prerequisites that are described in Prepare to Sideload Apps.

The device user needs administrator rights on the device to execute the Application Center client.

About this task

Installing APPX packages through Application Center is done by a process called sideloading. As part of Windows 8.1 Update, sideloading is enabled for all Windows 8.1 Pro devices that are part of an Active Directory domain, which matches the current behavior of Windows 8.1 Enterprise. If you use either of those product versions and the device is part of an Active Directory domain, you have no concerns about sideloading keys or activating sideloading.

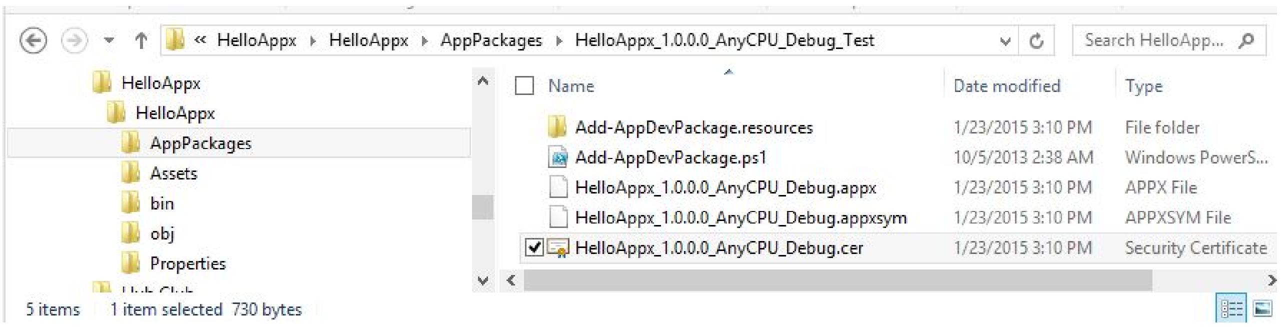
When you develop a Windows Store application, Microsoft Visual Studio automatically generates a self-signed certificate and uses it to code sign the application package. To be able to install the application later by using Application Center, you must import this certificate into the “Trusted Root Certification Authorities” store of the “Local Machine”. Importing the certificate is a manual procedure.

Note: Manual installation of a certificate is only required for the development phase, because APPX code signing relies on a self-signed certificate generated by Microsoft Visual Studio. In production, your APPX file must be signed by a genuine certificate purchased from a recognized root certificate authority.

Procedure

The first step of this procedure tells you how to install the certificate before you can install the application through Application Center.

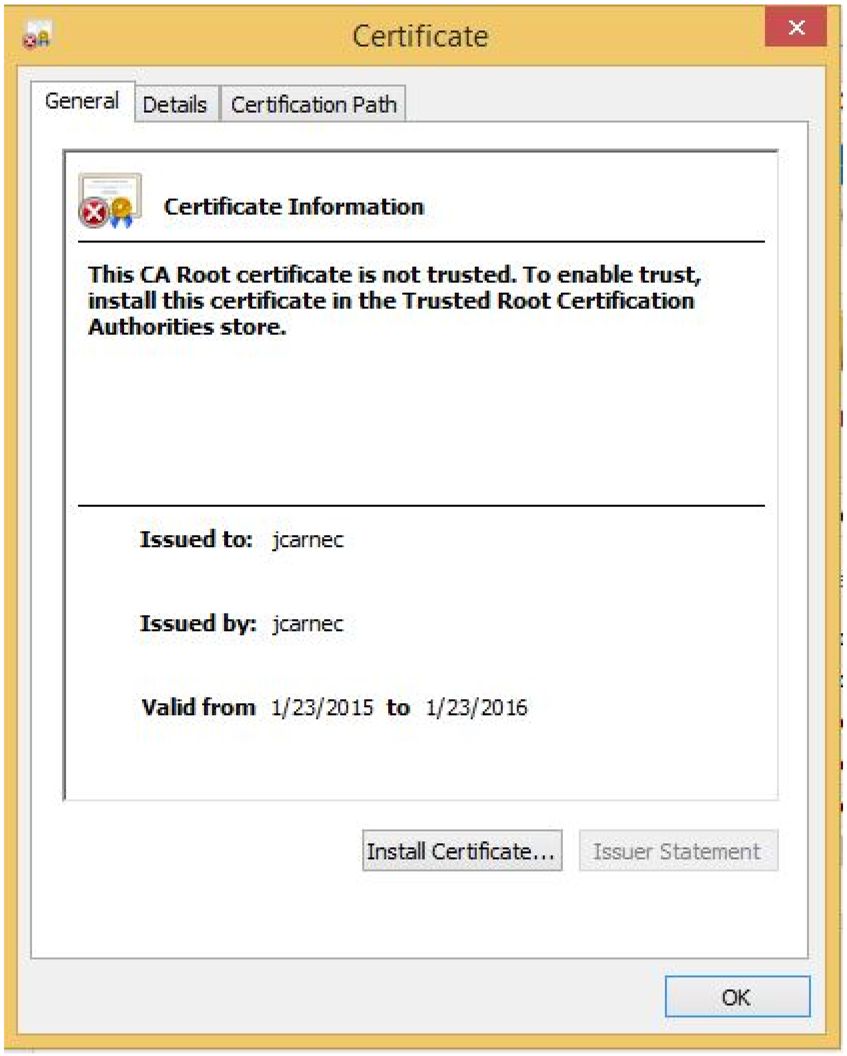
1. Import this certificate into the “Trusted Root Certification Authorities” store of the “Local Machine”.
   a. After you have generated an APPX file by using Visual Studio, place this file in your file system. In the folder of the APPX file, you can see a certificate (`.cer`) file that contains the self-signed certificate that you must import.


« HelloAppx ▸ HelloAppx ▸ AppPackages ▸ HelloAppx_1.0.0.0_AnyCPU_Debug_Test
Search HelloApp...
HelloAppx
HelloAppx
AppPackages
Assets
bin
obj
Properties
5 items
1 item selected 730 bytes

Name	Date modified	Type
Add-AppDevPackage.resources	1/23/2015 3:10 PM	File folder
Add-AppDevPackage.ps1	10/5/2013 2:38 AM	Windows PowerS...
HelloAppx_1.0.0.0_AnyCPU_Debug.appx	1/23/2015 3:10 PM	APPX File
HelloAppx_1.0.0.0_AnyCPU_Debug.appxsym	1/23/2015 3:10 PM	APPXSYM File
HelloAppx_1.0.0.0_AnyCPU_Debug.cer	1/23/2015 3:10 PM	Security Certificate

Figure 13-38. Certificate file in the application package folder

   b. To open the certificate, double-click the CER file.
   c. Click **Install Certificate**.



This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

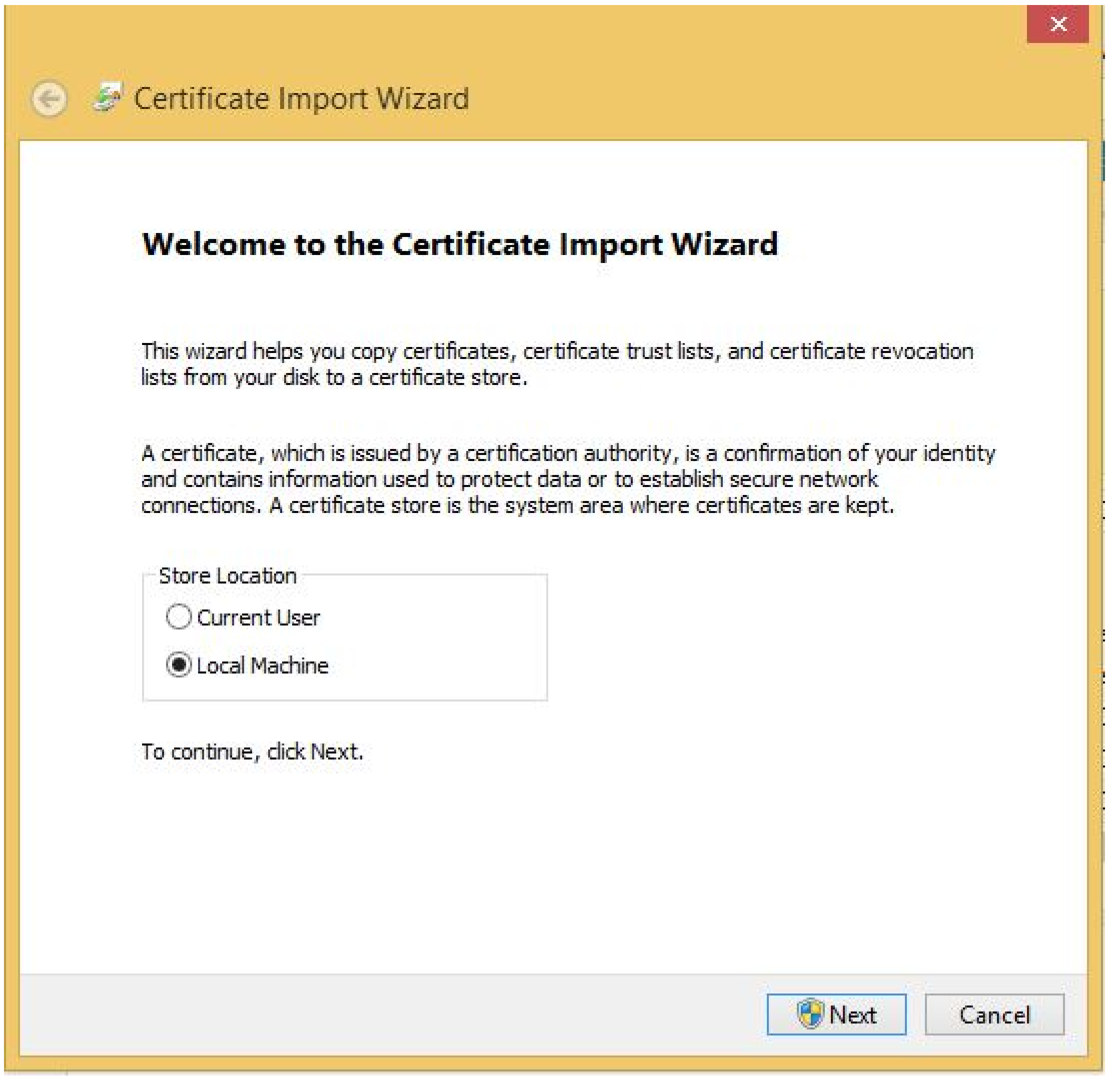
Issued to: jcarnec

Issued by: jcarnec

Valid from 1/23/2015 to 1/23/2016

Figure 13-39. General information about the certificate

d. Select “Local Machine” and click **Next**.



Welcome to the Certificate Import Wizard

This wizard helps you copy certificates, certificate trust lists, and certificate revocation lists from your disk to a certificate store.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

Store Location

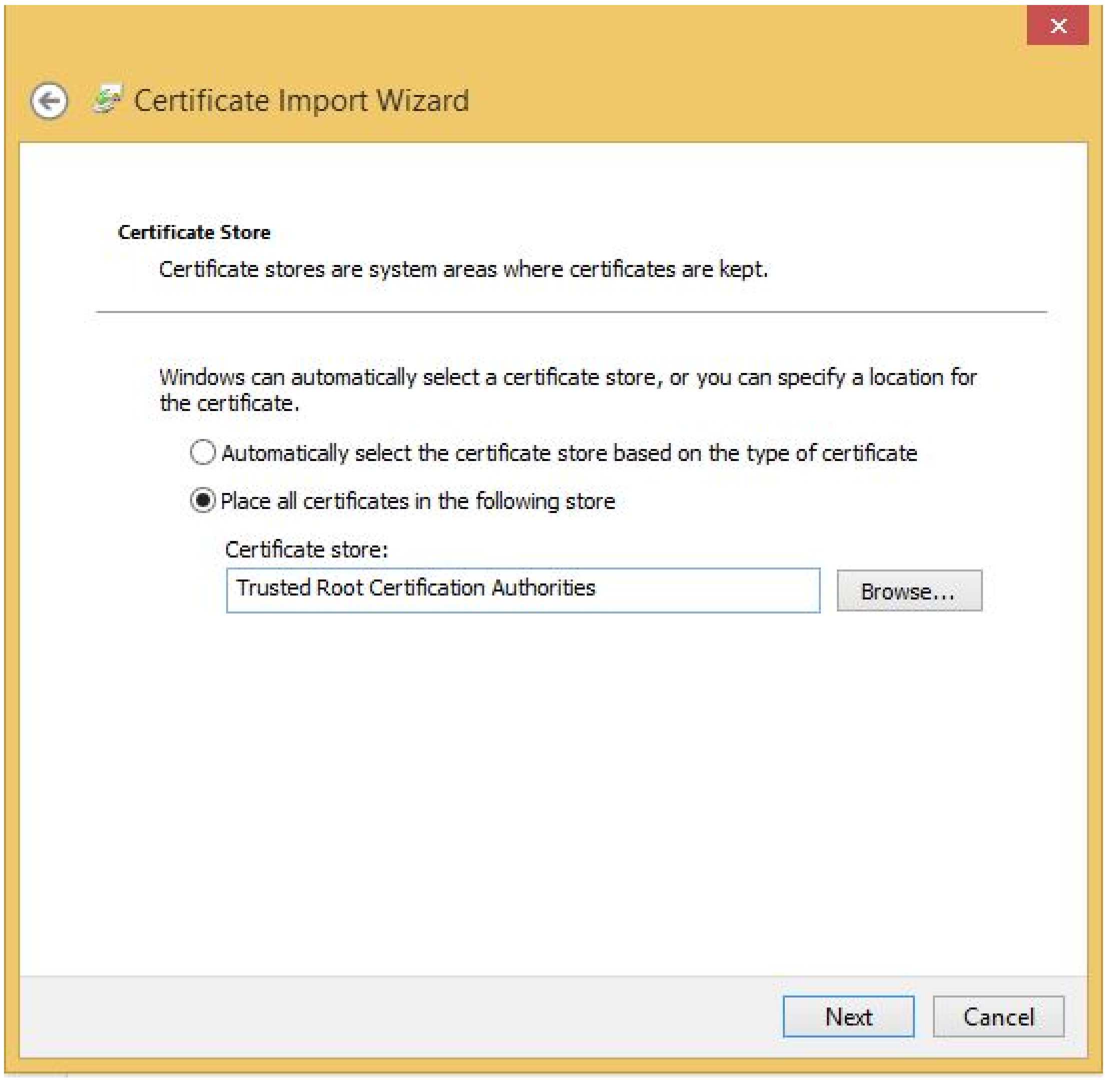
○ Current User

● Local Machine

To continue, click Next.

Figure 13-40. Specifying the local machine in the Certificate Import Wizard

e. Select “Place all certificate in the following store” and then browse to select “Trusted Root Certification Authorities”.



Certificate Import Wizard

Certificate Store

Certificate stores are system areas where certificates are kept.

Windows can automatically select a certificate store, or you can specify a location for the certificate.

( ) Automatically select the certificate store based on the type of certificate

(●) Place all certificates in the following store

Certificate store:

Trusted Root Certification Authorities

Figure 13-41. Placing the certificate in “Trusted Root Certification Authorities”

f. Click **Next** and then **Finish**. The successful import of the certificate should be confirmed.

The following steps describe how to perform the installation of a Windows Store application on a Windows device by using Application Center.

2. Log in to the Application Center mobile client for Windows Store applications.
3. Select the application that you want to install to access its details.

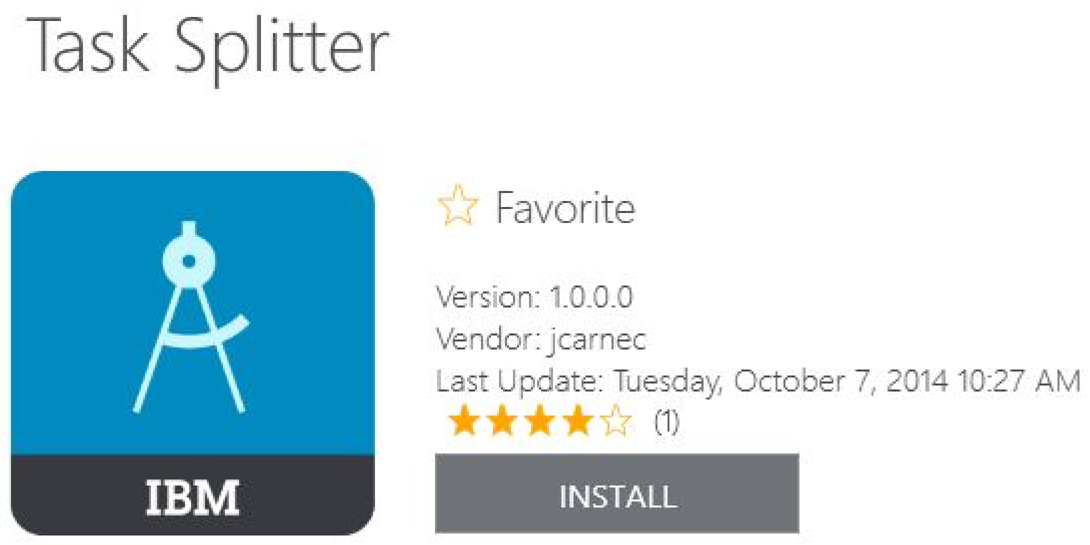


Figure 13-42. Details view for installing a Windows Store app

4. To install the application, tap **Install**. If the application is already installed and other versions are available, you can decide to update to a later version or to revert to a previous version.

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.

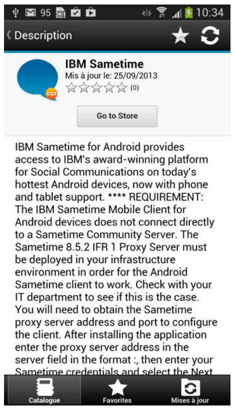
About this task

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 13-24. You can install these applications through the mobile client on your compatible device.

Links to Android applications stored in Google play and to iOS applications stored in Apple iTunes are listed in the application list on the device along with the binary files of private applications created within your enterprise.

Procedure

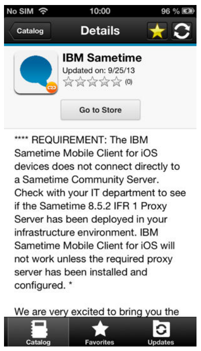
1. Select an application stored in a public app store from the application list to see the application details. Instead of **Install**, you see **Go to Store**.
2. Tap **Go to Store** to open Google play or Apple iTunes.



IBM Sametime
Mis à jour le: 25/09/2013
(0)

IBM Sametime for Android provides access to IBM's award-winning platform for Social Communications on today's hottest Android devices, now with phone and tablet support. **** REQUIREMENT: The IBM Sametime Mobile Client for Android devices does not connect directly to a Sametime Community Server. The Sametime 8.5.2 IFR 1 Proxy Server must be deployed in your infrastructure environment in order for the Android Sametime client to work. Check with your IT department to see if this is the case. You will need to obtain the Sametime proxy server address and port to configure the client. After installing the application enter the proxy server address in the server field in the format :, then enter your Sametime credentials and select the Next

Figure 13-43. Accessing an application in Google play from the mobile client on the device



IBM Sametime
Updated on: 9/25/13
(0)

**** 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 13-44. Accessing an application in Apple iTunes from the mobile client on the device

3. Follow the usual procedure of the public app store to install the application.

Removing an installed application

You can remove an application that is installed on your mobile device.

Procedure

1. Start the removal procedure that is valid for the operating system of your device.
   - **Android:** See the procedure in step 2.
   - **iOS**: You can remove applications only from the iOS Home screen, and not through the Application Center client. Use the normal iOS procedure for removing an application.
   - **Windows Phone**: You can remove applications only from the Windows Phone Home screen, and not through the Application Center client. Use the normal Windows Phone procedure for removing an application.
   - **Windows Store**: You can remove applications either from the Application Center mobile client or from the Windows home screen.
2. **Android only**: Remove an application from an Android device.
   a. In the **Details** view of any version of the application, tap **Uninstall**. The **Uninstall** button appears in the **Details** view only when a version of the application is installed. You are requested to confirm that the application version is to be uninstalled.

   b. Tap **Uninstall** to uninstall the application version or **Cancel** to notify the application that the uninstallation command has been canceled.

Showing details of a specific application version

You can show the details of the selected application version on any supported device.

About this task

You can show the details of the selected version of an application by following the appropriate procedure for an Android or iOS phone or tablet, a Windows Phone device, or a Windows device.

Procedure

1. Show details of a specific application version on a mobile device by selecting the appropriate procedure for your device.
   - A Windows Phone, Android, or iOS phoneA phone; see step 2.
   - A Windows device; see step 3
   - A tablet; see step 4.
2. **Windows Phone, Android, iOS only**: Show details of a specific application version on a Windows Phone, Android, or iOS phone.
   a. Tap **Select a version** to navigate to the version list view.

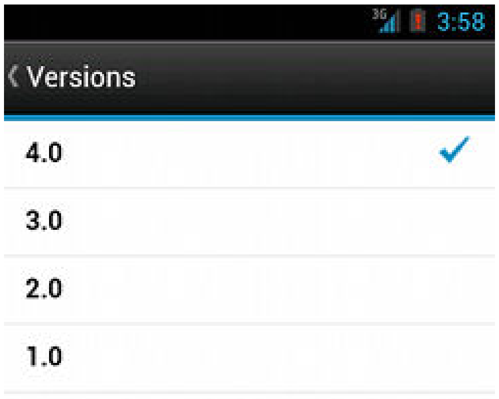


Figure 13-45. Specific version of an application selected in the list of versions on a Windows Phone, Android, or iOS phone

   b. Tap the appropriate version of the application. The **Details** view is updated and shows the details of the selected application version.
3. **Windows only**: Show details of a specific Windows Store application version on a Windows device. If more than one version is available for the Windows Store application, then you can select which version that you want to install.
   a. Tap the appropriate version of the application. The **Details** view is updated and shows the details of the selected application version.
4. **Tablet devices only**: Show details of a specific application version on a tablet.
   a. Tap **Select version**.
   b. In the menu, select the appropriate version of the application. The **Details** view is updated and shows the details of the selected application version.

Updating an application

You can update an application that is installed on your device if a new version is available in the Application Center.

About this task

Follow this procedure to make the latest versions of favorite and recommended apps available on your device. Applications that are marked as favorites and that have an updated version are listed in the **Updates** view. The applications that are marked as recommended by the Application Center server administrator are also listed in the **Updates** view, even if they are not favorites.

If a more up-to-date version of an installed application is available on the server, it is listed under **Update or Recommended**.

Procedure

1. In the **Updates** view, navigate to the **Details** view.
2. In the **Details** view, select a newer version of the application or take the latest available version.
3. **Android and Windows 8 Universal**: On Android and Windows 8 Universal devices, tap **Update**.
4. **iOS only**: On iOS devices, tap **Install latest**..
5. Follow the appropriate application installation procedure.
   - "Installing an application on an Android device" on page 13-66
   - "Installing an application on an iOS device" on page 13-68
   - "Installing an application on a Windows Phone device" on page 13-70
   - "Installing a Windows Store application on a Windows device" on page 13-72

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. This feature is supported for iOS, Android, and Windows Phone.

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.

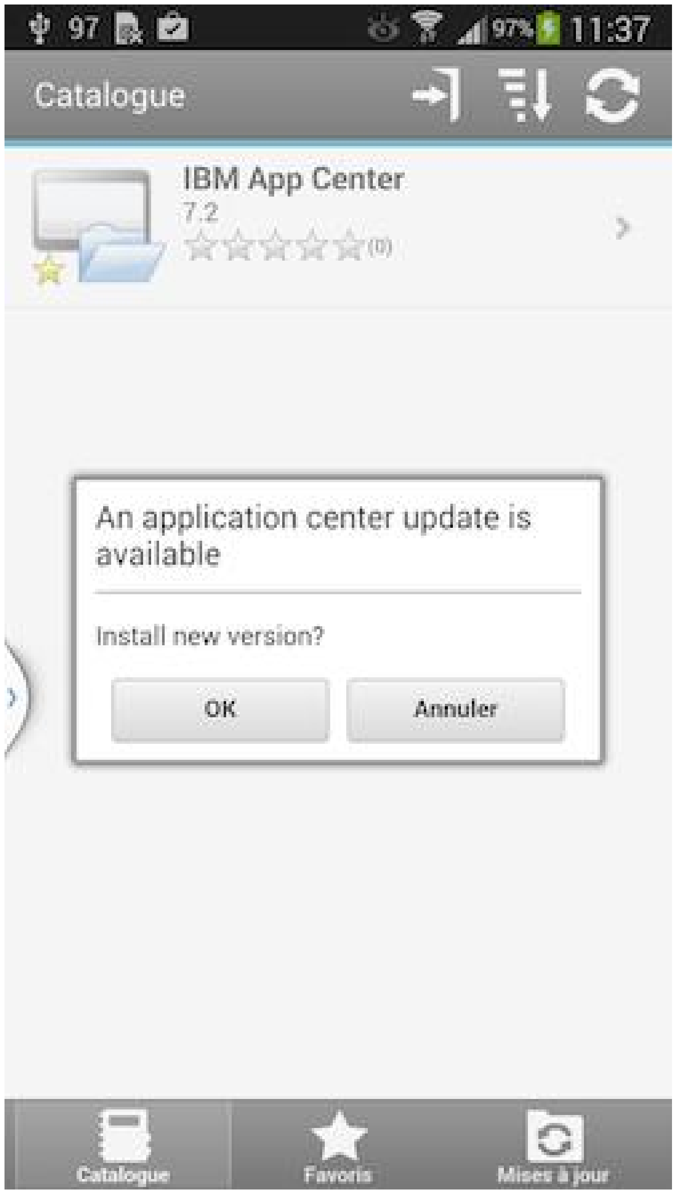
If a later version of the application is detected, you are requested to download and install the later version.

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.

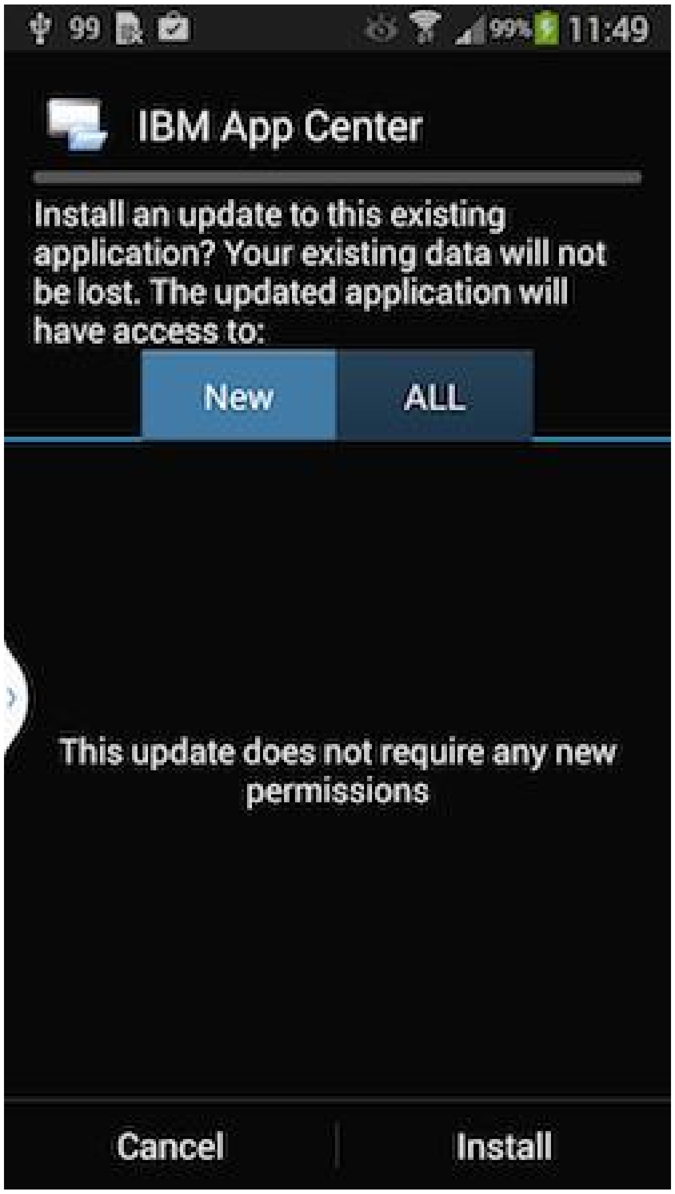


An application center update is available

Install new version?

Figure 13-46. Detection of a later version of the client application available on the server

2. Tap **Install** to install the later version of the application.


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 13-47. Confirm installation of the updated version of the application

3. Tap **Open** to start the updated application.

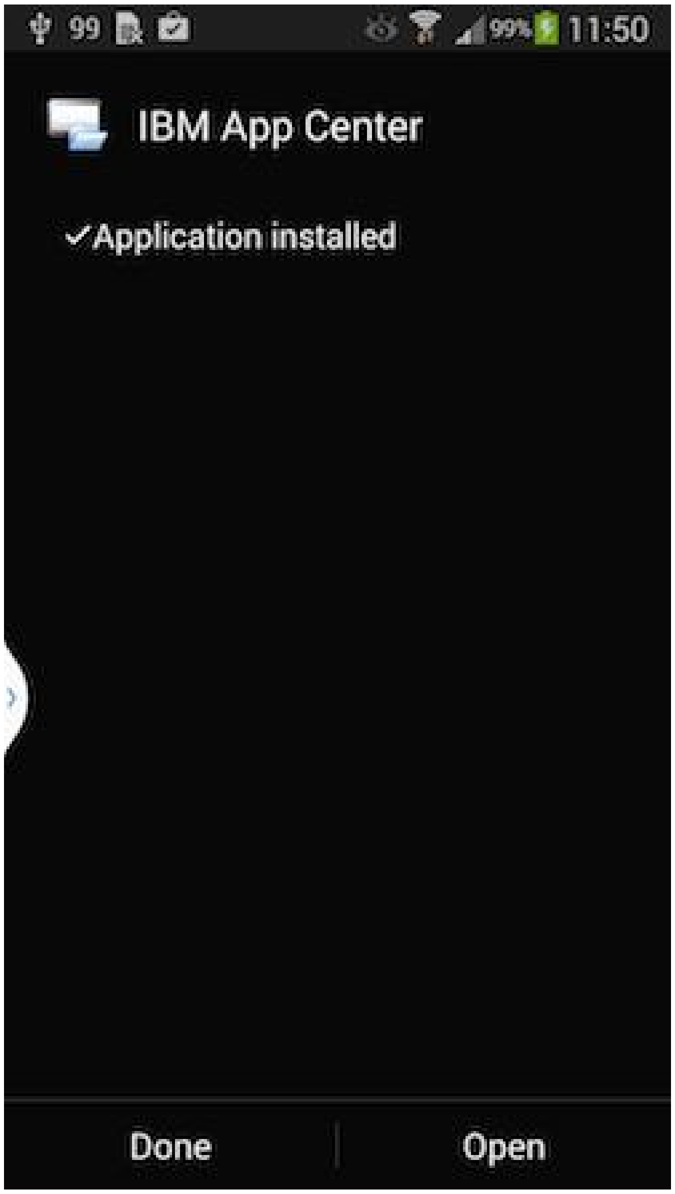
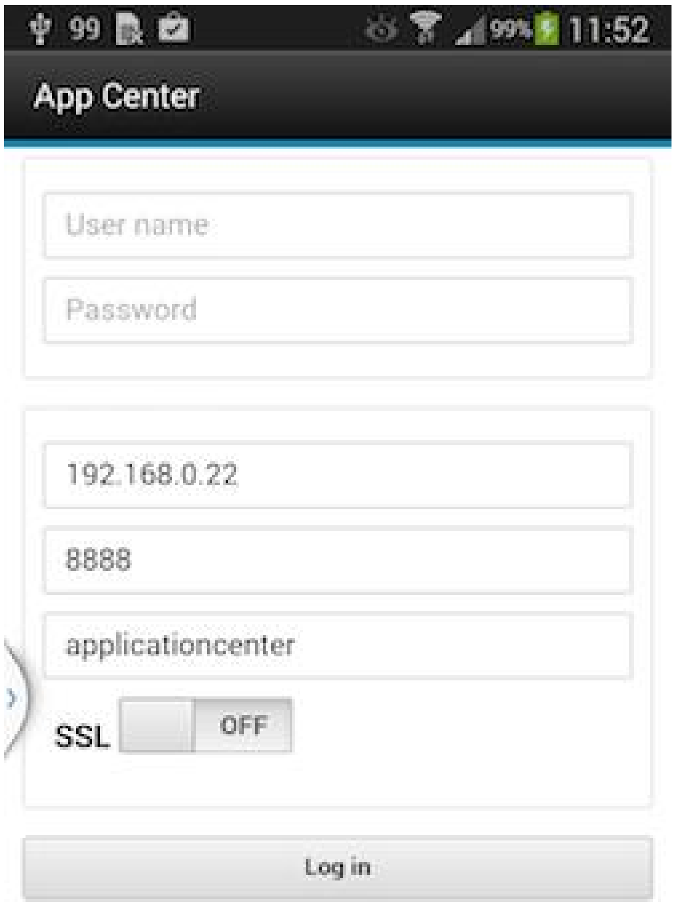


Figure 13-48. Starting the updated application

Results

You must log in to the updated version of the application to run it.



192.168.0.22

8888

applicationcenter

Figure 13-49. 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.
- On iOS, the new Application Center client must be signed with the same provisioning profile as the old client.
- On Android, the new Application Center client must have the same signature as the old client.
- On Windows Phone, the new Application Center client must be signed with the same company account 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 13-79 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 13-6 for information about how to disable reverting to earlier versions of an application.

On Android

If the installed version of the Android operating system is earlier than 4.2.2, tap **Revert**.

If the installed version of the Android operating system is 4.2.2 or later, you must uninstall the current version before you can install the earlier version.

Then, follow the procedure documented in "Installing an application on an Android device" on page 13-66.

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 13-68.

On Windows Phone

Tap **Revert**. Follow the procedue documented in "Installing an application on a Windows Phone device" on page 13-70.

Marking or unmarking a favorite app

Mark your favorite apps or unmark an app to have it removed from the favorites list.

An application marked as a favorite on your device indicates that you are interested in this application. This application is then listed in the list of favorite apps to make locating it easier. This application is displayed on every device belonging to you that is compatible with the application. If a later version of the app is available in the Application Center, the application is listed in the **Updates** view.

To mark or unmark an application as a favorite app, tap the Favorites icon ★ in the header of the **Details** view.

An installed application is automatically marked as a favorite app.

Submitting a review for an installed application

You can review an application version that is installed on your mobile device; the review must include a rating and a comment.

About this task

You can submit a review of an application version only 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**.
   - On Android phones and tablets, tap **Review version X**.
2. Enter a nonzero star rating.

   On mobile devices with touchscreens, tap 1 to 5 stars 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. On Android, iOS, and Windows Phone devices, the review details are visible in the list.

Viewing reviews of a specific version

The **Details** view always shows the details of a specific version. On a phone, the reviews are for that version.

In the **Details** view of an application version:

On a Windows Phone, Android, or iOS phone
: Tap **View Reviews** to navigate to the **Reviews** view.

On a tablet
: Tap **Reviews** *xx*, where *xx* is the displayed version of the application.

Viewing reviews of all versions of an application

In the **Details** view of an application version:

On a Windows Phone, Android, or iOS phone
: Tap **View Reviews** to navigate to the **Reviews** view. Then, tap the settings icon. ⚙, tap **All versions**, and confirm the selection.

On a tablet
: Tap **All Reviews**.

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 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), 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 WebSphere Application Server administrative console.
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 packages and their corresponding detail level. This example enables logging for IBM MobileFirst Platform Foundation, 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`, not to `SystemOut.log` or to `SystemErr.log`.

What to do next

For more information, see Configuring Java logging using the administrative console.

Enabling logging and tracing in WebSphere Application Server Liberty

You can set the logging levels and the output file for tracing operations for Application Center on the Liberty 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, 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"/>
```

In this example, multiple entries of a package and its logging level are separated by a colon (:).

The traces are sent to a file called `trace.log`, not to `messages.log` or to `console.log`.

For more information, 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, 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 information, 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 JNDI properties are applicable to the web application for Application Center services (`applicationcenter.war`).

Table 13-3. JNDI property settings for controlling trace output

Property settings	Setting	Description
`ibm.appcenter.logging.formatjson`	`true`	By default, this property is set to `false`. Set it to `true` to format JSON output with blank spaces, for easier reading in log files.
`ibm.appcenter.logging.tosystemerror`	`true`	By default, this property is set to `false`. Set it to `true` to print all log messages to system error in log files. Use the property to turn on logging globally.

Table 13-3. JNDI property settings for controlling trace output (continued)

Property settings	Setting	Description
`ibm.appcenter.openjpa.Log`	`DefaultLevel=WARN, Runtime=INFO, Tool=INFO, SQL=TRACE`	This setting prints all the generated SQL statements to the log files.

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 JSONStore" on page 7-144
- "Troubleshooting a corrupted login page (Apache Tomcat)" on page 13-20
- "Troubleshooting push notification problems" on page 7-265
- "Troubleshooting an error when an application or an adapter is pushed to a MobileFirst Server" on page 7-233
- "Troubleshooting Analytics and Logger" on page 11-49
- "Stale data after creating or deleting apps from MobileFirst Operations Console" on page 6-194

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.

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 15-2 "C" on page 15-2 "D" on page 15-4 "E" on page 15-4 "F" on page 15-4 "G" on page 15-5 "H" on page 15-5 "I" on page 15-5 "J" on page 15-5 "K" on page 15-6 "L" on page 15-6 "M" on page 15-7 "N" on page 15-7 "O" on page 15-8 "P" on page 15-8 "R" on page 15-9 "S" on page 15-9 "T" on page 15-10 "U" on page 15-11 "V" on page 15-11 "W" on page 15-11 "X" on page 15-11

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.

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.

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.

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

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

Note: Some topics of this documentation describe how IBM MobileFirst Platform Foundation integrates with third-party products. IBM does not support these third-party products. For information about how IBM supports the integration of IBM MobileFirst Platform Foundation with these third-party products, see Support statement for the IBM MobileFirst Platform Foundation family of products, under "*Other configurations*".

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

Trademarks

Terms and conditions for product documentation

Applicability

Personal use

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

Special characters

--debug option 7-20
--no-color option 7-20
--version option 7-20
-ddebug option 7-20

A

Access Control List (ACL)
  Application Center 6-238
  Application Center for WebSphere Application Server V8 6-240
  configuring Tomcat for LDAP authentication 6-246
  configuring user authentication for MobileFirst Server administration 6-166
  enabling with LDAP for WebSphere Application Server Liberty 6-243
access for users and groups
  Application Center 6-238, 6-242
access token
  OAuth security model 7-279
access tokens
  generation 7-303, 7-305
  how to obtain tokens 7-252
  OAuth 7-303, 7-305
  security for push notification clients 7-251
accessibility
  Application Center 2-10
  CLI 7-20
  configuration 2-10
  installation 2-10
  MobileFirst Analytics 2-10
  MobileFirst Server 2-10
  of web application consoles 2-10
accessibility features for this product 2-10
ACL
  See Access Control List (ACL)
ACL management
  JNDI properties for Application Center 6-261
Ad Hoc Distribution 13-2
adapter
  debugging 7-230
  resources
    protecting 7-271, 7-272
  testing 7-230
adapter accessmobile clientsnon-mobile clients 7-232
adapter structure
  Java 7-189
  JavaScript 7-189
adapter-descriptor file 7-219, 7-222
  Java 7-194
  JavaScript 7-206
adapter-maven-api 7-189
adapter-maven-plugin 7-189
adapter.xml file 7-206
  adapter element 7-194
  displayName attribute 7-194
adapters 7-192
  See also HTTP adapters
  anatomy 7-204
  audit logs 6-194
  benefits 7-204
  building 7-196, 7-197, 7-214, 7-215
  changes from V6.2 to V8.0.0 5-1
  configuring 7-196
    security checks 7-298
  creating 7-196
  deploying 7-196, 7-198, 7-214, 7-216
  deploying between environments 10-2
  deploying or updating to a production environment 10-2
  descriptor
    <securityCheckDefinition> element 7-294, 7-295
  developing 7-196, 7-214
  exporting and importing by using the REST API 10-11
  exporting and importing from the MobileFirst Operations Console 10-12
  getting started 7-9
  HTTP 7-219
  Java
    protecting resources 7-272
  Java, overview 7-192
  JavaScript
    protecting resources 7-272
  Maven 7-189
  mfpadm Ant task, commands 10-30
  mfpadm program commands 10-57
  overview 7-187
  pulling 7-196, 7-199, 7-214, 7-217
  pushing 7-196, 7-199, 7-214, 7-217
  SecurityCheck interface 7-291
  SecurityCheckConfigurtation interface 7-291
  SQL 7-222
  what's new in 8.0 3-10
adapters API 7-203
adaptersa
  Maven projects 5-51
  migrating 5-51
  upgrading 5-51
additional resources 4-1
administering applications
  messages
    defining in multiple languages 10-19
administration service
  REST API 8-7
administrator role
  in the Application Center console 13-18
administrator role *(continued)*
  to execute the Application Center mobile client 13-2
  user authentication for MobileFirst Server administration 6-166
alerts 11-30, 11-33
analytics 6-198, 8-270, 11-1, 11-2, 11-3, 11-12, 11-13, 11-14, 11-15, 11-19, 11-20, 11-21, 11-22, 11-23, 11-25, 11-26, 11-28, 11-29, 11-30, 11-33, 11-35, 11-36, 11-37, 11-38, 11-39, 11-40, 11-41, 11-42, 11-43, 11-44, 11-45, 11-46, 11-47, 11-48, 11-49
  introduction 11-1
  product main features 2-1
  SDK 11-35
Analytics
  enabling or disabling data collection from the MobileFirst Operations Console 11-23
analytics console 2-1
android 7-62
Android
  adding mobile applications to Application Center 13-22
  applications
    registering 7-59
    registering using MobileFirst Operations Console 7-61
    registering using MobileFirst Platform CLI 7-59
  deploying the mobile client in Application Center 13-12
  developing native applications 7-1, 7-24, 7-52
  installing and running the mobile client in Application Center 13-6
  installing the client 13-52
  preparation for the mobile client 13-8
  prerequisites for using the mobile client 13-6
  push notification from Application Center 13-12
  removing applications 13-78
  specific platform requirements for Application Center 13-2
  submitting reviews of installed applications 13-86
  updating applications in Application Center 13-80
Android application, new 7-53, 7-56, 7-57
Android applications
  getting started 7-9
  migrating with Gradle 5-26
  migration scenarios for push notification 5-62
  native
    preparing environment 7-53
    setting up environment 7-53
  properties 13-27
  pulling client logs 11-24

Android applications *(continued)*
  reverting 13-85
Android devices
  pre-installed root certificates 6-257
  reviews of application versions 13-86
  showing details of a specific application version 13-79
  views in the Application Center client 13-63
Android examples 7-183
Android projects
  upgrading native 5-25
Ant files
  Applying a fix pack 6-111
Ant program
  required version 10-23
Ant tasks 6-204, 6-304
  See also mfpadm Ant task
  application servers 6-318
  configuring application servers 6-293, 6-318
  configuring databases 6-268
  creating database tables for MobileFirst Server 6-73
  creating DB2 database for MobileFirst Server 6-75
  creating MySQL database for MobileFirst Server 6-78
  creating Oracle database for MobileFirst Server 6-76
  for IBM MobileFirst Platform Foundation installation 6-275, 6-287, 6-309
  for uploading or deleting an application 13-49
  install server farm 6-141
  installation of MobileFirst Server in command line mode 6-22
  installing MobileFirst Server components 6-110
  reference 6-318
  sample configuration files 6-318
  to administer MobileFirst applications 10-23
  to deploy Application Center console and services 6-206
  to install MobileFirst Analytics 11-10
Apache CXF JAR files
  known limitation 3-25
Apache Tomcat
  configuring to avoid MySQL, DB2, and Oracle timeout issues 6-192
  installing MobileFirst Analytics 11-6
  prerequisites for installing MobileFirst Server 6-100
  topologies 6-85, 6-88
Apache Tomcat server
  manual configuration 6-222
API
  See also REST API
  client
    challenge handlers 7-306
    ChallengeHandler 7-306
    OAuth 7-306
    security 7-306
    WLAuthorizationManager 7-306
    WLChallengeHandler 7-306
API *(continued)*
  client *(continued)*
    WLClient / WL.Client / WorklightClient 7-306
    WLResourceRequest 7-306
  deprecated and discontinued server-side APIs 5-14
  MobileFirst client SDKs 7-8
  server side
    SecurityCheck interface 7-291
    SecurityCheckConfiguration interface 7-291
API reference 8-1
app config 7-20
app crashes 11-46
app stores
  See application stores
App Transport Security (ATS)
  building applications for a test or production environment 10-4
  enforcing TLS-secure connections in iOS apps 7-46, 7-127
app usage 11-42
Apple
  Ad Hoc Distribution 13-2
Apple iTunes
  adding applications toApplication Center catalog 13-24
Apple provisioning profiles 13-2
Apple Push Notification Service (APNs)
  push notification for iOS devices 7-255
Apple watchOS 2 7-48
application
  configuration 7-20
application authenticity 7-282
Application Center 6-191, 6-204, 6-222
  access for users and groups 6-240
  adding applications from public app stores 13-24
  adding mobile applications to the repository 13-22
  administration console 13-18
  after EAR file deployment 6-231
  application version numbers 13-2
  concept 13-1
  configuring 6-198
  configuring Derby manually on Tomcat 6-214
  configuring LDAP authentication for Apache Tomcat 6-246
  configuring LDAP authentication for WebSphere Application Server Liberty 6-242
  configuring Liberty profile for Oracle manually 6-219
  configuring Secure Sockets Layer (SSL) 6-257
  configuring SSL 6-257
  configuring the endpoint of application resources for WebSphere Application Server 6-253
  configuring the endpoint of application resources for WebSphere Application Server Liberty 6-255
Application Center *(continued)*
  configuring the Java EE security roles on WebSphere Application Server 6-234
  configuring the Java EE security roles on WebSphere Application Server Liberty 6-236
  configuring the server scheduler 13-13
  configuring Tomcat for DB2 manually 6-210
  configuring Tomcat for LDAP authentication 6-246
  configuring WebSphere Application Server for DB2 manually 6-209
  configuring WebSphere Application Server for Derby manually 6-212
  configuring WebSphere Application Server Liberty manually
    after EAR file deployment 6-229
  configuring WebSphere Application Server manually 6-224, 6-231
  deploying EAR file and configuring the application server manually 6-229
  deploying MobileFirst Server to IBM PureApplication 9-44
  deploying the console and services with Ant tasks 6-206
  deploying the mobile client 13-12
  deploying WAR files 6-222
  general architecture 13-4
  installer
    prerequisites for building 13-6
  installing 6-198
  installing manually 6-207
  installing with Ant tasks 6-304
  introduction 13-1
  JNDI properties 6-261
  JNDI properties for controlling trace output 13-88
  known limitations 3-25
  LDAP and WebSphere Application Server V8 6-238
  logging and tracing in Tomcat 13-88
  logging and tracing in WebSphere Application Server full profile 13-87
  logging and tracing in WebSphere Application Server Liberty profile 13-88
  managing and installing self-signed CA certificates in a test environment 6-260
  manual configuration of DB2 6-208
  mobile client 13-51
    importing and building the project 13-8
    launched automatically 13-27
    prerequisites 13-6
  preparing for installation 13-6
  presentation 2-4
  product main features 2-1
  removing applications 13-78
  repository 13-18
  roles 13-18

Application Center *(continued)*
  sefl-signed certificates not supported 6-257
  setting log and trace parameters 13-87
  setting up your Derby database manually 6-211
  setting up your MySQL database manually 6-215
  settting up your DB2 database manually 6-207
  specific platform requirements 13-2
  submitting reviews of installed applications 13-86
  updating applications 13-80
  updating production apps 10-14
Application Center Access Control List
  Virtual Member Manager 6-238
Application Center client
  configuring for push notification 13-12
Application Center console
  configuring LDAP authentication for WebSphere Application Server V8 6-238
  signing out 13-45
  troubleshooting a corrupted login page on Safari 13-21
  troubleshooting a corrupted login page on Tomcat 13-20
Application Center database
  Ant tasks 6-204
  configuring 6-204
  creating 6-204
application configurations
  See configurations
application descriptors
  for Cordova applications 7-102
application resources
  configuring the endpoint 6-253, 6-255
application server 5-1
  Apache Tomcat 6-101
  configuring
    Ant task 6-318
    Ant tasks 6-293
    reference 6-318
  JMX connection 6-101, 6-103
  WebSphere Application Server Liberty profile 6-103
application servers
  prerequisites 6-100
  setting logging and tracing for Application Center 13-87
  supported for server farm configuration 6-139
application status
  in MobileFirst Operations Console with token licensing 10-20
application store 2-1
application stores
  adding applications from 13-24
application versions
  exporting and importing by using the REST API 10-11
  exporting and importing from the MobileFirst Operations Console 10-12
applications
  See also native applications
  See also recommended applications
  adding toApplication Center catalog from public app stores 13-24
  administering 10-1
  administering through Ant 10-23
  administering through the command line 10-47
  authenticity
    configure 7-284
    enable 7-282
    mfp-app-authenticity-tool 7-282
    tool 7-282
  authenticity validation and tool 7-282
  building for a test or production environment 10-4
  configuring
    enableSSO 7-301
    mandatoryScope 7-276
    maxTokenExpiration 7-304
    scopeElementMapping 7-277
    security checks 7-300
  configuring FIPS 140-2 10-79
  configuring push notification for updates 13-13
  deploying and managing
    what's new in 8.0 3-10
  deprecated and discontinued server-side APIs 5-14
  developing and publishing
    product main features 2-1
  disabling remotely 10-17
  exporting and importing by using the REST API 10-11
  exporting and importing from the MobileFirst Operations Console 10-12
  getting started 7-9
  installing on an iOS mobile device 13-55, 13-68
  license validation 10-83
  mandatory scope 7-265, 7-276
  maximum access-token expiration period 7-304
  messages
    multiple languages 10-19
  mfpadm Ant task commands 10-34
  mfpadm program commands 10-61
  mobile
    administrator messages 10-18
    blocking access to protected resources 10-17
    management 10-15
  NuGet 7-65
  overview 7-2
  properties 13-27
  rating 13-86
  registering
    Android applications 7-59
    Android applications using MobileFirst Operations Console 7-61
    Android applications using MobileFirst Platform CLI 7-59
    Cordova 7-107
applications *(continued)*
  registering *(continued)*
    Cordova applications using MobileFirst Operations Console 7-109
    Cordova applications using MobileFirst Platform CLI 7-107
    iOs 7-37
    iOS applications using MobileFirst Operations Console 7-38
    iOS applications using MobileFirst Platform CLI 7-37
    web applications 7-80
    web applications using MobileFirst Operations Console 7-81
    web applications using MobileFirst Platform CLI 7-80
    Windows 7-68
    Windows applications using MobileFirst Operations Console 7-70
    Windows applications using MobileFirst Platform CLI 7-68
  registering to a production environment 10-6
  removing from Application Center 13-78
  reverting 13-85
  reviews of application versions 13-86
  samples 7-25
  security configurations
    LTPA-based single sign-on (SSO) predefined security check 7-289
  setting up your environment for development 7-65
  sharing through Application Center 13-1
  showing details of a specific version 13-79
  stale data after creation or deletion from MobileFirst Operations Console 6-194
  submitting a review 13-86
  subscribing to tags for push notification 7-257
  transferring configurations from one server to another 10-9
  updating in Application Center 13-80
  upgrading from V6.2 to V8.0.0 5-1
  uploading or deleting by using an Ant task 13-49
  web
    See web applications
  what's new 3-1
  Windows 8, sideloading 13-2
applications, Cordova 7-87
apps
  deploying between environments 10-2
  production apps
    best practices 10-14
  updating in production 10-14
architecture
  of Application Center 13-4
  push notification 7-250
ARM-based tablets
  .exe files not executed 13-2

artifacts
  See server-side artifacts
asymmetric deployment
  Liberty collective 6-91
  WebSphere Application Server Network Deployment 6-94
audit logs
  for adapters 6-194
  for MobileFirst Operations Console 6-194
  of administration operations 10-21
authentication
  changes in the security model 5-1
  configuring Tomcat for LDAP authentication 6-246
  configuring user authentication for MobileFirst Server administration 6-166
  LDAP
    for Apache Tomcat 6-246
authenticity
  See application authenticity
authenticity data
  transferring server-side artifacts to another server 10-7
authenticity validation 7-239

B

back end
  method for push notification architecture 7-250
  polling method 7-250
back-end connections
  product main features 2-1
backup 11-19
Bitcode
  build options 7-48
blocked application status
  token licensing 10-20
Bluemix
  containers 9-1
Bouncy Castle cryptographic library
  known limitation 3-25
broadcast notifications
  sending to the device 7-258
  unsubscription 7-256
browsers
  Safari 13-21
building
  adapters 7-197, 7-215

C

C#
  developing native applications for Windows 7-65
CA certificates
  file name extensions 6-260
capacity 11-3
capturing data 10-78
certificate authority 7-185
certificate pinning 7-185
  building applications for a test or production environment 10-4
certificates
  See also CA certificates
  self-signed
    to configure SSL 10-3
changes 3-1
changesinterim fixes 3-16
circuit breakers 11-22
class loaders
  loading policy 6-224, 6-231
class loading policies
  after EAR file deployment 6-231
  configuring WebSphere Application Server for Application Center manually 6-224, 6-231
CLI 7-13, 7-14, 7-17, 7-19
  accessibility 7-20
  getting started 7-22
  prerequisite software 7-14
CLI modes 7-19
CLI, installing 7-15
client API
  acquiring the SDK from the IBM MobileFirst Platform Operations Console 7-26
  JavaScript API for web and Cordova applications 7-73, 7-76, 7-79
  JavaScript web analytics API 7-73, 7-75, 7-76, 7-79
  web SDK 7-73, 7-75
    acquiring 7-75
    adding to the application 7-76
    initializing the SDK 7-79
client applications
  development environments 7-8
  MobileFirst SDKs 7-8
client log profiles
  configuring from the MobileFirst Operations Console 11-24
client property file 7-62, 7-72
client property files
  for native iOS applications 7-41
client SDK 11-35, 11-37, 11-42
client-side
  Android 8-5, 8-6
  API 8-5, 8-6
  C# 8-6
  Java 8-5, 8-6
  push API 8-6
  Windows 10 Universal Windows Platform (UWP) 8-6
  Windows 8 Universal 8-6
client-side API
  iOS 8-5
  Objective-C and Apple Swift language 8-5
client-side push API
  iOS 8-5
cloud
  deploying to the cloud as Liberty for Java Cloud Foundry application 9-4
  deploying to the cloud in an IBM Container 9-1
  deployment of MobileFirst Server 9-44
Cloudant
  MFData CloudantToolkit no longer supported 5-1
cluster 11-19, 11-20, 11-21
CocoaPods, developing native application with 7-29, 7-34
command 7-13
command line 7-17
  installing MobileFirst Server 6-22
command-line 7-13
command-line interface 7-14, 7-22
  defining server 7-16
  getting started 7-22
  prerequisite software 7-14
command-line interface modes 7-19
command-line summary 7-17
command-line tools
  mfpadm 10-47
Company Hub
  Windows Phone 8 applications 13-2
components
  IBM MobileFirst Platform Foundation 2-4
confidential clients
  configuring 7-264
  JNDI properties for MobileFirst Server administration service 6-174
configuration 11-12, 11-14, 11-15
  Apache Tomcat 6-101
  application 7-20
  JMX connection 6-101, 6-103, 6-104
  sample files for MobileFirst Analytics 6-319
  server farm 6-144
  WebSphere Application Server 6-104
  WebSphere Application Server Liberty profile 6-103
  WebSphere Application Server Network Deployment 6-104
configuration API 7-203
configuration files
  for Cordova applications 7-102
  for server farms 6-139
configurations
  transferring by using mfpdev to a different environment 10-7
configuredatabase
  Ant task 6-268
configuring
  adapters 7-196
  application servers 6-150
  data sources 6-190
  JNDI properties of
    MobileFirst Server web applications 6-171
  keystore 7-316
  server farms 6-139
Configuring
  DB2 HADR seamless failover 6-191
configuring for MobileFirst Server administration 6-170
configuring the web resources checksum test 7-20
connection
  Apache Tomcat 6-153
  MobileFirst Server 6-153, 6-154, 6-156

connection *(continued)*
  Rational License Key Server 6-153, 6-154, 6-156
  WebSphere Application Server 6-156
  WebSphere Application Server Liberty profile 6-154
  WebSphere Application Server Network Deployment 6-156
connectionPolicy
  attributes 7-209, 7-213
  elements 7-209, 7-213
connections
  handling stale connections 6-192
console 11-13
constraints
  of MobileFirst Analytics 6-84
  of MobileFirst Server components 6-84
  of MobileFirst Server push service 6-99
containers
  deploying to the cloud in an IBM Container 9-1
  overview 9-1
  package structure and contents 9-2
context root 6-293
Converting, Cordova application 5-43
Cordova 7-87
  application resources 7-102
  applications
    registering 7-107
    registering using MobileFirst Operations Console 7-109
    registering using MobileFirst Platform CLI 7-107
  creating 7-89
  Eclipse 7-115
  platforms and plug-ins 2-4
  template 7-88
Cordova app, adding MobileFirst Platform Foundation 7-91
Cordova application, converting 5-43
Cordova applications 7-87
  building for a test or production environment 10-4
  changes from V6.2 to V8.0.0 5-1
  converted from a hybrid application 5-41
  developing 7-83
  getting started 7-9
  known limitations 3-25
  prerequisites 7-84
  what's new 3-1
  WKWebView plug-in 7-134
Cordova apps
  start-up flow 7-129
  viewing project 7-128
  Windows 7-128, 7-129
Cordova, migrating to 5-38
crash 11-45, 11-48
creating
  adapters 7-196
  Cordova 7-89
creating Cordova applications 7-87
credentials
  to Application Center mobile client 13-51
credentials *(continued)*
  to connect to Application Center from the Login view 13-61
  to upload an application 13-49
Cross Origin Resource Sharing (CORS)
  JNDI properties for MobileFirst Server administration service 6-174
cross-platform applications
  pulling client logs 11-24
Crosswalk WebView interface
  developing Cordova applications 7-83
CSS files
  as Cordova application resources 7-102
custom
  icons 7-104
  images 7-104
  splash screens 7-104
custom analytics 11-23, 11-35, 11-48
custom charts 11-25, 11-26, 11-28, 11-29, 11-30, 11-44, 11-49
custom data 11-36
custom nodes 9-108
customizing
  Cordova application resources 7-102

D

data
  enabling or disabling collection from the MobileFirst Operations Console 11-23
data source
  configuring for MobileFirst Server applications 6-64
data sources 6-191
  configuring 6-190
database
  configuring with an external database 9-91
  creating tables for MobileFirst Server manually 6-67
  requirement for MobileFirst Server 6-65, 6-66
  users and privileges 6-64
database creation
  for MobileFirst Server manually 6-67
  for MobileFirst Server with Ant tasks 6-73
  for MobileFirst Server with the Server Configuration Tool 6-70
database user
  privileges to access MobileFirst Server database tables 6-65, 6-66
databases
  configuring
    Ant task 6-268, 6-318
  handling MySQL stale connections 6-192
  internal runtime databases 6-315
  MobileFirst Server components 6-63
datasource JDBC properties
  sample Ant files for MobileFirst Server 6-112
DB2
  configuring manually for Application Center on Tomcat 6-210
  configuring WebSphere Application Server manually for Application Center 6-209
  creating database for MobileFirst Server manually 6-67
  creating database for MobileFirst Server with Ant tasks 6-75
  creating database for MobileFirst Server with Server Configuration Tool 6-71
  database requirement for MobileFirst Server 6-65
  in combination with Apache Tomcat, WebSphere Application Server, or WebSphere Application Server Liberty profile 6-192
  installation of MobileFirst Server in command line mode 6-22
  installation of MobileFirst Server in graphical mode 6-5
  setting up your database manually for Application Center 6-207
  stale connections 6-192
DB2 database creation
  for MobileFirst Server manually 6-67
  for MobileFirst Server with Ant tasks 6-75
  for MobileFirst Server with Server Configuration Tool 6-71
DB2 SQL Error 6-190, 6-252
debug options 7-20
default charts 11-43, 11-47
delete, Ant task 13-49
dependencies 7-24
dependency manager
  iOS 7-29, 7-34
deployer role
  user authentication for MobileFirst Server administration 6-166
deploying
  adapters 7-198, 7-216, 10-2
  apps 10-2
  updated apps 10-14
deprecated API 3-17
  application server side 5-14
deprecated features 3-17
Derby
  configuring manually for Application Center on Tomcat 6-214
  configuring WebSphere Application Server manually for Application Center 6-212
  setting up the database manually for Application Center 6-211
Derby databases
  not supported by server farms 6-139
developing
  Cordova apps 7-122, 7-123, 7-124, 7-129, 7-130
    Windows 7-128, 7-129
  Crosswalk plugin 7-133
developing applications 7-65
  product main features 2-1
development environment 2-1, 5-1

Development Kit 7-9
Development Kit, installing 7-10
development of applications
  getting started 7-9
devices
  configuring for push notification 13-12
  mfpadm Ant task commands 10-42
  mfpadm program commands 10-68
  search limitations 3-25
  synchronization with application log profile 11-24
DevOps
  what's new 3-10
direct mode 7-19
direct update
  client apps 7-239
  implementing 7-239
Direct Update 7-237
  app versions 7-240
  CLI 7-236
  customize 7-244
  default UI 7-240
  development and production environments 7-235
  full vs. incremental 7-235
  keystore 7-235, 7-316
  lifecycle 7-236
  MobileFirst Operations Console 7-236
  overview 7-235
  public key setting 7-20
  secure and non-secure 7-235
  user interface 7-244
discontinued features
  API 3-19
distribution structure
  of MobileFirst Server 6-61
Docker containers
  See containers
downloading
  IBM MobileFirst Platform Foundation 2-10

E

EAR file
  to deploy Application Center manually 6-229
Eclipse 7-187, 7-189
  Cordova 7-115
  supported versions 2-7
editable properties
  of mobile applications 13-27
endpoints
  of application resources, configuring 6-253, 6-255
environment variables
  to configure the Application Center server scheduler 13-13
environments
  production 10-2
  QA 10-2
  test 10-2
error
  deploying with Application Center console 6-252
error *(continued)*
  deploying with MobileFirst Operations Console 6-190
  transaction log full 6-190, 6-252
Event source based notification
  upgrading from 5-55
event-source-based notification
  not supported in V.8.0.0 5-1
examples 7-182
external
  resources
    MobileFirst Java Token Validator 7-274
    protecting 7-274

F

failure 7-148
farm
  See server farms
favorite applications
  push notification 13-12
feature table 2-10
feature-platform matrix 2-10
federal 10-74
Federal Desktop Core Configuration 10-74, 10-75
Federal Information Processing Standards (FIPS)
  security standards 10-75, 10-79
Federal Information Processing Standards (FIPS) 140-2
  configuring for existing applications 10-79
federated registries
  configuring LDAP ACL management for Liberty 6-243
file extensions
  ignored during web resources checksum test 7-20
files
  of native API applications for iOS, copying 7-27, 7-31, 7-32, 7-48, 7-51
FIPS 140-2 10-78
  enabling 10-77
  known limitations 3-25
  not available in Cordova applications 5-41
firewalls
  JNDI properties for Application Center 6-261
fix packs
  by Ant files 6-111
  by Server Configuration Tool 6-110
floating license 2-8
framework
  JavaScript adapter 7-204

G

getting started 4-1, 7-22
  with the MobileFirst Operations Console 7-6
globalization
  known limitations 3-25
glossary 15-1
Google Play
  adding applications toApplication Center catalog 13-24
Gradle
  migrating Android projects 5-26
Gradle, developing native application with 7-53, 7-56, 7-57

H

heartbeat rate
  for server farm nodes 6-149
homogeneous server farms
  supported 6-139
HTTP
  Strict Transport Security standards 6-174
HTTP adapter
  connectionPolicy 7-209
  elements 7-209
HTTP adapter XML file 7-209
HTTP adapter XML file structure 7-209
HTTP adapters
  and WebSphere Application Server SSL configuration 6-197
  example 7-219
  procedures 7-219
  SSL 7-222
HTTPS
  keystore 7-316
HTTPS protocol
  JNDI properties for MobileFirst Server administration service 6-174
hybrid applications
  changes from V6.2 to V8.0.0 5-1
  converting into a Cordova application 5-41
  migration scenarios for push notification 5-56
  Objective-C client-side API 8-5
hybrid apps, migrating to Cordova 5-38

I

IBM Installation Manager
  administrator mode 6-42
  installation of MobileFirst Server in command line mode 6-22
  installation of MobileFirst Server in graphical mode 6-5
  record response files
    MobileFirst Server installation 6-50
  user mode 6-42
IBM Installation Manager command line
  installing MobileFirst Server 6-45
IBM Installation Manager Install wizard
  installing MobileFirst Server 6-43
IBM MobileFirst Platform Application Center
  accessibility 2-10
IBM MobileFirst Platform Command Line Interface (CLI)
  presentation 2-4
  registering applications
    Android 7-59

IBM MobileFirst Platform Command Line Interface (CLI) *(continued)*
registering applications *(continued)*
Cordova 7-107
iOS 7-37
web 7-80
Windows 7-68
IBM MobileFirst Platform Foundation 7-13, 8-4
IBM MobileFirst Platform Operations Console
accessibility 2-10
adapters
security 7-294, 7-298
application
authenticity 7-282, 7-284
predefined LTPA-based SSO security check 7-289
applications
security 7-276, 7-277, 7-282, 7-284, 7-300, 7-301, 7-304
client SDKs download 7-26
MobileFirst Server keystore configuration 7-316
registering applications
Android 7-61
Cordova 7-109
iOS 7-38
web 7-81
Windows 7-70
IBM WebSphere DataPowerusing as the OAuth authorization server 7-314
icons
as Cordova application resources 7-102
custom 7-104
images for Cordova apps 7-104
images
custom 7-104
images for Cordova apps
custom 7-104
in the Application Center client
views in the client 13-63
index.html 7-132
installanalytics
Ant task 6-309
installation 6-198, 11-2
Ant tasks 6-275, 6-287, 6-309
installing MobileFirst Analytics on 11-4, 11-6, 11-7
of MobileFirst Analytics by using Ant tasks 11-10
of MobileFirst Server 6-4, 6-5, 6-22
running IBM Installation Manager 6-41
server farm with Ant tasks 6-141
server farm with the Server Configuration Tool 6-140
using IBM Installation Manager command line 6-45
using IBM Installation Manager Install wizard 6-43
Installation
MobileFirst Server components with Ant Tasks 6-110
installing
MobileFirst Platform Pattern 9-48
installing *(continued)*
MobileFirst Server to an application server 6-100
token licensing
overview 6-150
Installing the IBM MobileFirst Platform Foundation Developer Kit 7-10
installing, CLI 7-15
installing, Development Kit 7-10
installingMobileFirst Server 6-2
installmobilefirstadmin
Ant task 6-275
installmobilefirstpush
Ant task 6-287
installmobilefirstruntime
Ant task 6-293
IntelliJ 7-187, 7-189
developing Cordova applications 7-83
interactive mode 7-19
invalid server farm configurations 6-139
Ionic
developing Cordova applications 7-83
iOS
adding mobile applications to Application Center 13-22
applications
registering 7-37
registering using MobileFirst Operations Console 7-38
registering using MobileFirst Platform CLI 7-37
client property file for native applications 7-41
credentials stored in keychain 13-61
deploying the mobile client in Application Center 13-12
developing native applications 7-1, 7-24, 7-27, 7-126
installing and running the mobile client in Application Center 13-6
preparation for the mobile client 13-8
prerequisites for using the mobile client 13-6
push notification from Application Center 13-12
removing applications 13-78
specific platform requirements for Application Center 13-2
SSL mandatory to connect to Application Center 13-61
submitting reviews of installed applications 13-86
troubleshooting push notification problems 7-265
updating applications in Application Center 13-80
iOS application, new 7-29, 7-34
iOS applications
configuring the Application Center server with SSL 6-257
enforcing TLS-secure connections 7-46, 7-127
getting started 7-9
migration scenarios for push notification 5-69
iOS applications *(continued)*
native
preparing environment 7-27
setting up environment 7-27
Objective-C client-side API 8-5
Objective-C client-side push API 8-5
properties 13-27
pulling client logs 11-24
reverting 13-85
WKWebView plug-in 7-134
iOS devices
installing applications 13-68
installing the client 13-55
iOS 9, establishing trust on a provisioning file 13-55, 13-68
pre-installed root certificates 6-257
reviews of application versions 13-86
showing details of a specific application version 13-79
views in the Application Center client 13-63
iOS examples 7-182
iOS hybrid applications
Objective-C client-side API 8-5
iOS projects
upgrading native 5-17, 5-23, 5-33, 5-44, 5-45

J

JAR files 7-24
Java
protecting external resources 7-271, 7-274
Java adapter
configuration 7-194
Java adapters
benefits 7-192
Java EE
configuring security roles for Application Center on application servers 6-234, 6-236
Java EE level
sample Ant files 6-112
WebSphere Application Server Liberty 6-112
Java EE security roles
configuring user authentication for MobileFirst Server administration 6-166
configuring WebSphere Application Server for MobileFirst Server administration 6-169
configuring WebSphere Application Server Liberty profile for MobileFirst Server administration 6-169
Java Management Extensions (JMX)
JNDI properties for MobileFirst Server administration service 6-174
Java Persistence API (JPA)
JNDI properties 6-174
Java Runtime Environment (JRE)
truststores 6-197
Java server-side API 7-203
JavaScript 8-4
modules
loading 7-76

JavaScript *(continued)*
  RequireJS 7-76
JavaScript adapter
  configuration 7-206
JavaScript adapter framework 7-204
JavaScript adapter XML file
  attributes 7-194, 7-206
  elements 7-194, 7-206
  sub-elements 7-194, 7-206
JavaScript adapters
  benefits 7-204
  global variables 7-218
  gzipped responses 7-218
  overview 7-204
  response threshold 7-218
JavaScript examples 7-184
JavaScript files
  as Cordova application resources 7-102
JavaScript server-side API
  logger 7-225
  server 7-225
JAX-RS service 7-200
JCE policy files
  to troubleshoot iOS push notification problems 7-265
JDBC driver class 7-222
JMX connection configuration
  WebSphere Application Server 6-104
  WebSphere Application Server Network Deployment 6-104
JNDI properties
  configuring LDAP ACL management for Liberty 6-243
  configuring the endpoint of application resources for WebSphere Application Server 6-253
  configuring the endpoint of application resources for WebSphere Application Server Liberty 6-255
  defining environment variables for Application Center server scheduler 13-13
  entries for MobileFirst runtime in production 6-183
  for Application Center 6-261
  for Application Center logs and traces 13-87
  for controlling trace output for Application Center 13-88
  for MobileFirst Server administration service 6-174
  for MobileFirst Server live update service 6-182
  for MobileFirst Server push service 6-186
  for WebSphere Application Server Network Deployment topology 6-94
  IBM Cloudant database 6-174
  MobileFirst projects, configuring 6-171
  of MobileFirst Server web applications 6-171
  runtime 6-183
  sample Ant files for MobileFirst Server 6-112
JNDI properties *(continued)*
  to configure a server farm 6-144
  to install a mobile client on an iOS mobile device 13-55
  to install applications on an iOS mobile device 13-68
JSON objects
  formatting, JNDI property 6-174
JSONStore 7-134, 7-181, 7-182, 7-183, 7-184
  advanced 7-169
  analytics 7-180
  API 7-140
  concurrency 7-173
  enabling 7-139
  error codes 7-148
  errors 7-147
  examples 7-151
  Federal Information Processing Standards (FIPS) 10-75
  general terminology 7-137
  Java 7-162
  JavaScript 7-151
  multiple user support 7-171
  Objective-C 7-158
  overview 7-134, 7-144
  performance 7-171
  security 7-169, 7-170, 7-171
  SQLCipher 7-171
  SQLite 7-171
  supported architectures 3-25
  sync 7-174
  troubleshooting 7-144

K

keyboard shortcuts
  for web application consoles 2-10
keychain
  iOS credentials 13-61
Keytool
  for self-signed certificates 10-3
Kit, Development 7-9
  installing 7-10
known issues 3-25, 14-1
known limitations 3-25

L

language sensitivity
  limitation on search 3-25
language settings 7-20
LDAP
  See also Lightweight Directory Access Protocol (LDAP)
  authentication for WebSphere Application Server Liberty 6-242
  authentication for WebSphere Application Server V8 6-238
  configuring security with an external repository 9-86
  configuring Tomcat for authentication 6-246
  JNDI properties for Application Center 6-261
LDAP ACL management
  configuring for the Liberty profile 6-243
LDAP authentication
  for Apache Tomcat 6-246
Liberty
  deploying to the cloud as Liberty for Java Cloud Foundry application 9-4
Liberty collective
  topologies 6-91
Liberty profile
  configuring for Oracle for Application Center 6-219
  configuring Java EE security roles for Application Center 6-236
  configuring LDAP authentication 6-242
  configuring manually for Application Center 6-208, 6-222
    after EAR file deployment 6-229
  configuring the endpoint of application resources 6-255
  configuring WebSphere Application Server Liberty profile 6-169
  logging and tracing in Application Center 13-88
  Oracle
    configuring Liberty profile manually for Application Center 6-219
libraries
  of native API applications for iOS, copying 7-27, 7-31, 7-32, 7-48, 7-51
  Rational Common Licensing 10-83
License Key Server
  registering applications to a production environment 10-6
License terms 10-80
license tracking 10-80
licenses
  validation 10-83
licensing
  perpetual license 2-8
  token licensing 2-8
  traditional floating license 2-8
Lightweight Directory Access Protocol (LDAP)
  Application Center on WebSphere Application Server V8 6-238
Lightweight Directory Access Protocol (LDAP)
  Application Center on WebSphere Application Server V8 6-240
lightweight third-party authentication
  See LTPA
limitations 3-25
  to deploying MobileFirst Server to IBM PureApplication 9-44
Linux OS
  stopping the MobileFirst Development Server 7-13
log profiles, client side
  configuring from the MobileFirst Operations Console 11-24
logger 11-1, 11-37, 11-49
  introduction 11-1

logging
  See also audit logs
  file location 6-194
  JNDI properties 6-174
  monitoring tools 6-194
logging and tracing
  in Tomcat 13-88
  in WebSphere Application Server full profile 13-87
  in WebSphere Application Server Liberty profile 13-88
  setting parameters for Application Center 13-87
login page
  of the Application Center console, troubleshooting for Safari 13-21
  of the Application Center console, troubleshooting for Tomcat 13-20
Login view
  Application Center
    connecting from the Login view 13-61
  connecting to the Application Center 13-61
LOGSECOND 6-190, 6-252
LPTA
  known limitations on tokens 3-25
LTPA 7-285
  configuring user authentication for MobileFirst Server administration 6-166
  single sign-on (SSO) predefined security check 7-285
    configuring 7-289

M

Mac OS
  stopping the MobileFirst Development Server 7-13
management 11-19
management console 2-1
manual configuration
  after EAR file deployment 6-231
  configuring WebSphere Application Server for Derby sfor Application Center 6-212
  DB2 for Application Center on WebSphere Application Server Liberty profile 6-208
  DB2 for WebSphere Application Server manually for Application Center 6-209
  Oracle database 6-218
  setting up your DB2 database for Application Center 6-207
  setting up your Derby database Application Center 6-211
  setting up your MySQL database for Application Center 6-215
  Tomcat for DB2 for Application Center 6-210
  WebSphere Application Server for Application Center 6-224
    after EAR file deployment 6-231
  WebSphere Application Server Liberty profile for Application Center 6-222

manual configuration *(continued)*
    after EAR file deployment 6-229
  WebSphere Application Server profile for Application Center 6-231
manual configuration of WebSphere Application Server Liberty 6-222
manual installation
  Application Center 6-207
Manual installation
  configuring MobileFirst Operations Console on Tomcat 6-130
  configuring MobileFirst Operations Console on WebSphere Application Server 6-136
  configuring MobileFirst Operations Console on WebSphere Application Server Liberty 6-119
  configuring MobileFirst Operations Console on WebSphere Application Server Liberty collective 6-125
  configuringMobileFirst runtime on Tomcat 6-131
  configuringMobileFirst runtime on WebSphere Application Server 6-137
  configuringMobileFirst runtime on WebSphere Application Server Liberty 6-119
  configuringMobileFirst runtime on WebSphere Application Server Liberty collective 6-126
  configuringMobileFirst Server administration service on Tomcat 6-129
  configuringMobileFirst Server administration service on WebSphere Application Server 6-134
  configuringMobileFirst Server administration service on WebSphere Application Server Liberty 6-117
  configuringMobileFirst Server administration service on WebSphere Application Server Liberty collective 6-124
  configuringMobileFirst Server artifacts on Tomcat 6-132
  configuringMobileFirst Server artifacts on WebSphere Application Server 6-139
  configuringMobileFirst Server artifacts on WebSphere Application Server Liberty 6-121
  configuringMobileFirst Server artifacts on WebSphere Application Server Liberty collective 6-128
  configuringMobileFirst Server live update service on Tomcat 6-130
  configuringMobileFirst Server live update service on WebSphere Application Server 6-135
  configuringMobileFirst Server live update service on WebSphere Application Server Liberty 6-118
  configuringMobileFirst Server live update service on WebSphere Application Server Liberty collective 6-125

Manual installation *(continued)*
  configuringMobileFirst Server push service on Tomcat 6-132
  configuringMobileFirst Server push service on WebSphere Application Server 6-138
  configuringMobileFirst Server push service on WebSphere Application Server Liberty 6-120
  configuringMobileFirst Server push service on WebSphere Application Server Liberty collective 6-127
  MobileFirst Server on Tomcat 6-128
  MobileFirst Server on WebSphere Application Server Liberty 6-115
  MobileFirst Server on WebSphere Application Server Liberty collective 6-121
  MobileFirst Server to an application server 6-115
maven 7-187
Maven
  artifacts 7-189
  build goal 7-189
  build tool for adapters 5-1
  configpull goal 7-189
  configpush goal 7-189
  dependencies 7-189
  deploy goal 7-189
  goals 7-189
  plugin goal 7-189
  repositories 7-189
maven artifacts
  offline 7-24
Maven Central 7-189
maven internal repository 7-24
messages
  administrator
    adding locales 10-19
mfp-security-checks-base 7-189
mfpadm Ant task
  commands for adapters 10-30
  commands for apps 10-34
  commands for devices 10-42
  commands for general configuration 10-28
  commands for troubleshooting 10-44
  syntax, attribures, elements, XML format, output character set 10-24
  to administer MobileFirst applications 10-23
mfpadm program
  administering applications through the command line 10-47
  calling 10-48
  commands for adapters 10-57
  commands for applications 10-61
  commands for devices 10-68
  commands for general configuration 10-54
  commands for troubleshooting 10-71
mfpadm tool
  transferring an application configuration from one server to another 10-9
mfpdev 7-13, 7-17
  getting started 7-22

mfpdev app config 7-20
mfpdev command-line
  prerequisite software 7-14
migrating 5-1
migrating to Cordova 5-38
migration 5-1, 5-17, 11-12, 11-13
  deprecated and discontinued server-side APIs 5-14
  scenarios of migrating push notification for hybrid applications 5-56
  scenarios of migrating push notification for native Android applications 5-62
  scenarios of migrating push notification for native iOS applications 5-69
  scenarios of migrating push notification for Windows Universal applications 5-75
  V6.2 applications to V8.0.0 5-1
minification
  not supported in V.8.0.0 5-1
mobile applications 2-1
  See also applications
  adding to Application Center 13-22
Mobile Browser Simulator 7-109
  Cordova 7-109
  Overview 7-109
  previewing apps 7-109
mobile client
  Application Center component 13-51
  credentials 13-51
  deploying in Application Center 13-12
  installing and running in Application Center 13-6
  installing on an iOS mobile device 13-55
mobile client of Application Center
  importing and building the project 13-8
  prerequisites 13-6
mobile devices
  See also devices
  blocking access to protected resources 10-15
  tracking and managing 10-15
mobile-application management 10-15
  administrator messages 10-18
  administrator messages in multiple languages 10-19
  disabling application access to protected resources remotely 10-17
MobileFirst administration services
  heartbeat mechanism to detect unresponsive servers 6-149
MobileFirst Analytics 6-309
  constraints 6-84
  installing by using Ant tasks 11-10
  installing on Apache Tomcat 11-6
  installing on WebSphere Application Server 11-7
  installing on WebSphere Application Server Liberty 11-4
  sample configuration files 6-319
MobileFirst Development Server
  stopping 7-13
MobileFirst Java Token Validator 7-274
MobileFirst Node.js framework 7-275
MobileFirst Operations Console 7-282
  administering applications 10-1
  application authenticity 7-282
  audit logs 6-194
  checking the status of farm servers 6-149
  configuration details for Tomcat 6-130
  configuration details for WebSphere Application Server 6-136
  configuration details for WebSphere Application Server Liberty 6-119
  configuration details for WebSphere Application Server Liberty collective 6-125
  configuring client log profiles 11-24
  defining administrator messages in multiple languages 10-19
  defining scope mapping elements to security checks 7-263
  device management 10-15
  displaying an administrator message 10-18
  enabling or disabling Analytics data collection 11-23
  exporting and importing applications and adapters 10-12
  getting started with developing applications 7-9
  list of push notification tags 7-257
  mobile-application management 10-15
  occasional stale data, troubleshooting 6-194
  opening 7-13
  overview 7-6
  presentation 2-4
  remotely disabling application access 10-17
  search limitations 3-25
  what's new in 8.0 3-10
MobileFirst Platform Foundation Development Kit 7-9
MobileFirst Platform Foundation plug-ins for Cordova 7-91
MobileFirst Platform Foundation SDK for Cordova 7-91
MobileFirst Platform Pattern
  installing 9-48
  predefined templates 9-98
  script packages 9-108
MobileFirst projects
  configuring with JNDI properties 6-171
  upgrading 5-17
MobileFirst runtime
  configuration details for Tomcat 6-131
  configuration details for WebSphere Application Server 6-137
  configuration details for WebSphere Application Server Liberty 6-119
MobileFirst runtime *(continued)*
  configuration details for WebSphere Application Server Liberty collective 6-126
  topology constraints 6-84
MobileFirst runtime synchronization 9-76
MobileFirst Server 6-191, 7-265
  configuring Tomcat 6-170
  configuring user authentication 6-166
  deploying on IBM PureApplication System 9-44
  distribution structure 6-61
  handling MySQL stale connections 6-192
  IBM MobileFirst Platform Foundation System Pattern scripts 9-108
  installation 6-2
  installation prerequisites 6-39
  installing to an application server 6-100
  installing to an application server by using the Server Configuration Tool 6-105
  JNDI properties for administration service 6-174
  JNDI properties for live update service 6-182
  JNDI properties for push service 6-186
  logging 6-194
  mfpadm Ant task, commands for global configuration 10-28
  mfpadm program, commands for global configuration 10-54
  migration 5-1
  network flows 6-78
  overview 6-2
  running IBM Installation Manager 6-41
  server topologies 6-78
  Transport Layer Security v1.2 (TLS v1.2) 6-165
  upgrade 5-1
MobileFirst Server administration
  configuring WebSphere Application Server 6-169
  configuring WebSphere Application Server Liberty profile 6-169
MobileFirst Server administration service
  configuration details for Tomcat 6-129
  configuration details for WebSphere Application Server 6-134
  configuration details for WebSphere Application Server Liberty 6-117
  configuration details for WebSphere Application Server Liberty collective 6-124
  topology constraints 6-84
  transferring an application configuration from one server to another 10-9
MobileFirst Server artifacts
  configuration details for Tomcat 6-132

MobileFirst Server artifacts *(continued)*
configuration details for WebSphere Application Server 6-139
configuration details for WebSphere Application Server Liberty 6-121
configuration details for WebSphere Application Server Liberty collective 6-128
MobileFirst Server components
constraints 6-84
installing on Tomcat manually 6-128
installing on WebSphere Application Server Liberty manually 6-115
installing to an application server manually 6-115
network flows 6-78
MobileFirst Server live update service
configuration details for Tomcat 6-130
configuration details for WebSphere Application Server 6-135
configuration details for WebSphere Application Server Liberty 6-118
configuration details for WebSphere Application Server Liberty collective 6-125
topology constraints 6-84
MobileFirst Server push service
configuration details for Tomcat 6-132
configuration details for WebSphere Application Server 6-138
configuration details for WebSphere Application Server Liberty 6-120
configuration details for WebSphere Application Server Liberty collective 6-127
constraints 6-99
MobileFirst service
mfpadm Ant task commands for troubleshooting 10-44
mfpadm program commands for troubleshooting 10-71
MobileFirst Studio
migration 5-1
upgrade path 5-1
using MobileFirst Studio 7.1 3-25
modes, command-line interface 7-19
monitor role
user authentication for MobileFirst Server administration 6-166
monitoring
logging mechanism 6-194
product main features 2-1
multiple MobileFirst runtimes
topologies 6-99
MySQL
creating database for MobileFirst Server manually 6-69
creating database for MobileFirst Server with Ant tasks 6-78
creating database for MobileFirst Server with Server Configuration Tool 6-73
database requirement for MobileFirst Server 6-66
MySQL *(continued)*
in combination with Apache Tomcat, WebSphere Application Server, or WebSphere Application Server Liberty profile 6-192
setting up your database manually for Application Center 6-215
stale connections 6-192
troubleshooting MobileFirst Operations Console on Tomcat 8 6-194
MySQL database creation
for MobileFirst Server manually 6-69
for MobileFirst Server with Ant tasks 6-78
for MobileFirst Server with Server Configuration Tool 6-73

N

native Android projects
upgrading 5-25
native API applications
for iOS
copying files 7-27, 7-31, 7-32, 7-48, 7-51
native applications
developing 7-1, 7-24
developing for Android 7-52
for iOS 7-27, 7-126
for Universal Windows Platform (UWP) 7-65
for Windows 8 Universal 7-65
upgrading from V6.2 to V8.0.0 5-1
native iOS applications
client property file 7-41
native iOS projects
upgrading 5-17, 5-23, 5-33, 5-44, 5-45
native iOS, upgrading 5-19
native Windows application
upgrading 5-31
network flows
between MobileFirst Server components 6-78
new features 3-1
new featuresinterim fixes 3-16
node 11-20, 11-21
Node Package Manager (npm)
required software for Cordova applications 7-84
Node.js
MobileFirst Node.js framework 7-275
required software for Cordova applications 7-84
nodes, of server farms
lifecycle 6-149
notification
broadcast 7-256
notifications
tag-based, sending 7-258
npm
See Node Package Manager (npm)
NuGet
developing native applications 7-65, 7-66, 7-67

O

OAuth
overview 7-265
OAuth security model
access tokens 7-279
changes in the security model 5-1
JNDI properties for MobileFirst Server administration service 6-174
obfuscation
of password in mfpadm configuration file 10-48
object 8-4
Objective-C
client-side API for iOS 8-5
client-side push API for iOS 8-5
offline mode
product main features 2-1
One-Time URLs
controlled by a JNDI property 13-55
OpenJPA
See Java Persistence API (JPA) 6-174
operating systems
supported 2-7
supported by containers 9-1
operator role
user authentication for MobileFirst Server administration 6-166
Oracle
creating database for MobileFirst Server manually 6-68
creating database for MobileFirst Server with Ant tasks 6-76
creating database for MobileFirst Server with Server Configuration Tool 6-72
database requirement for MobileFirst Server 6-65
in combination with Apache Tomcat, WebSphere Application Server, or WebSphere Application Server Liberty profile 6-192
setting up your database manually 6-218
stale connections 6-192
Oracle database creation
for MobileFirst Server manually 6-68
for MobileFirst Server with Ant tasks 6-76
for MobileFirst Server with Server Configuration Tool 6-72
Oracle databases 6-220, 6-222
Apache Tomcat server 6-222
manual configuration 6-220, 6-222
WebSphere Application Server 6-220
overview 7-181
installing
token licensing 6-151
JavaScript adapters 7-204
Overview
MobileFirst Server 6-2

P

parent last
class loading policy 6-224, 6-231

patterns
  deploying MobileFirst Server to IBM PureApplication 9-44
Patterns
  presentation 2-4
phones
  reviews of application versions 13-86
  showing details of a specific application version 13-79
  submitting reviews of installed applications 13-86
platform limitations 6-150
preparing for developing Android native applications 7-53
preparing for developing iOS native applications 7-27
prerequisites
  Apache Tomcat 6-100
  application servers 6-100
  Cordova 7-84
  file system 6-104
  installing MobileFirst Server 6-39
  MobileFirst Server installation on an application server 6-104
  WebSphere Application Server 6-104
  WebSphere Application Server Liberty 6-103
  WebSphere Application Server Network Deployment 6-104
procedures
  HTTP adapters 7-219
  SQL adapters 7-222
product components 2-4
product overview 2-1
production
  JNDI environment entries for MobileFirst runtime 6-183
production environment 5-1
  deploying or updating adapters 10-2
  registering applications 10-6
production environments
  building applications 10-4
  transferring application configurations 10-7
production servers
  transferring server-side artifacts 10-7
profiles
  See also Ad Hoc Distribution
  See client log profiles
projects
  configuring with JNDI properties 6-171
properties 11-12
  of mobile applications 13-27
property files
  for native iOS applications 7-41
proxy settings
  for push notification 7-248
public app stores
  See application stores
public application stores
  adding applications from 13-24
public key 7-185, 7-235, 7-239
  direct update 7-20
pulling
  adapters 7-199, 7-217
PureApplication
  deploying MobileFirst Server, benefits and limitations 9-44
push notification
  Android 7-254
  architecture 7-248, 7-250
  broadcast 7-256
  configuring 7-259
  configuring for application updates 13-13
  getting started 7-251
  iOS 7-255
  mechanism 7-248
  migration scenarios for hybrid applications 5-56
  migration scenarios for native Android applications 5-62
  migration scenarios for native iOS applications 5-69
  migration scenarios for Windows Universal applications 5-75
  product main features 2-1
  proxy settings 7-248
  REST API 8-197
  scope mapping elements to security checks 7-263
  scopes 7-264
  security for clients 7-251, 7-252
  sending notifications to subscribers 7-262
  sending to the device 7-258
  setting up 7-254
  tag subscriptions 7-257
  tag-based notification 7-257, 7-258
  troubleshooting problems on iOS 7-265
Push notification
  upgrading to 5-55
push notifications
  sending
    using MobileFirst Operations Console 7-259
  Windows 7-256
push service
  changes from V6.2 to V8.0.0 5-1
pushing
  adapters 7-199, 7-217

Q

quick start
  on application development 7-9

R

Rational Common Licensing
  native library 6-160
  shared library 6-160
Rational Common Licensing native library
  token license validation 10-83
Rational License Key Server
  token license validation 10-83
  troubleshooting token licenses 6-160
recommended applications
  push notification 13-12
referrals
  supported or not 6-243
registration
  applications
    See applications, registering
relational databases
  IBM DB2 6-63
  MySQL 6-63
  Oracle 6-63
release notes 3-1, 3-25
  known limitations 3-25
removed features 5-4
required software 7-14
resources
  custom requests using WLAuthorizationManager 7-306, 7-307
    C# 7-313
    Java 7-309
    JavaScript 7-311
    Objective-C 7-308
  exporting and importing by using the REST API 10-11
  protecting 7-271
REST API 8-270
  administration service 8-7
  exporting and importing applications and adapters 10-11
  for the administration service
    what's new 3-3
  for the MobileFirst runtime 7-274
  push notification service 8-197
  runtime 8-270
  transferring an application configuration from one server to another 10-9
REST services
  administering applications 10-1
RESTful access
  JavaScript adapters 7-232
reverse proxies
  configuring MobileFirst Server 6-165
reverse proxy
  deployment of topologies of server farm and WebSphere Application Server Network Deployment 6-98
  JNDI properties for Application Center 6-261
reviews of application versions 13-86
reviews of applications 13-86
RFC 6797
  HTTP Strict Transport Security standards 6-174
role-based access 11-23
roles
  See also security
  for enabling or disabling data collection from the MobileFirst Operations Console 11-23
  for exporting and importing applications and adapters 10-11, 10-12
  mapping to users for Application Center Java EE security 6-234, 6-236
  mapping users 6-275, 6-287

roles *(continued)*
  user authentication for MobileFirst Server administration 6-166
root certificates
  pre-installed on Android and iOS devices 6-257
  self-signed CA certificates in an Application Center test environment 6-260
runtime
  internal databases 6-315
  REST API 8-270
runtime middleware 2-1
runtime synchronization limitation 9-77
runtimes
  exporting and importing applications and adapters by using the REST API 10-11
  exporting and importing applications and adapters from the MobileFirst Operations Console 10-12
  mfpadm Ant task, commands for global configuration 10-28
  mfpadm program, commands for global configuration 10-54

S

Safari browsers
  troubleshooting a corrupted Application Center login page 13-21
Sample Ant files
  modifications 6-112
sample response files
  installing MobileFirst Server 6-48
samples
  getting started with developing applications 7-9
  of MobileFirst applications 7-25
  to configure MobileFirst Analytics 6-319
scripts
  as Cordova application resources 7-102
SDK
  for Analytics 11-35
search
  limitations in MobileFirst Operations Console 3-25
Secure Socket Layer (SSL) configuration
  configuring in WebSphere Application Server, HTTP adapters 6-197
Secure Sockets Layer
  See SSL
security 7-265
  See also Java EE security roles
  authorization server 7-265
    WebSphere DataPower 7-265, 7-314
  challenge handlers 7-265, 7-306
  changes in the security model 5-1
  checks
    application authenticity 7-282, 7-284
    configure 7-284
security 7-265 *(continued)*
  checks *(continued)*
    predefined MobileFirst security checks 7-282, 7-284
    properties 7-284
  client API 7-306
  client certificate 7-279
  confidential clients 7-279
  configurations 7-297
  configuring Tomcat for LDAP authentication 6-246
  configuring user authentication for MobileFirst Server administration 6-166
  configuring with an external LDAP repository 9-86
  device single sign-on (SSO) 7-301
  enforcing TLS-secure connections in iOS apps 7-46, 7-127
  Federal Information Processing Standards (FIPS) 10-75
  for One-Time URLs 13-55
  for push notification clients 7-251, 7-252
  framework
    overview 7-265
  HTTP Strict Transport Security standards 6-174
  interfaces
    securityCheckDefinition 7-294
  Java EE security roles for Application Center, configuring 6-234, 6-236
  keystore 7-316
  mapping users to roles 6-275, 6-287
  OAuth 7-265, 7-314
    access tokens 7-265, 7-303, 7-305
    client API 7-306
    token response 7-305
    tokens 7-304, 7-316
    WLAuthorizationManager 7-307, 7-308, 7-309, 7-311, 7-313
  OAuth API 7-307
    C# 7-313
    Java 7-309
    JavaScript 7-311
    Objective-C 7-308
  OAuth scopes 7-265, 7-271, 7-272
    mandatory application scope 7-265, 7-276
    mapping 7-277
    scope elements 7-265, 7-277
  obfuscation of password in mfpadm configuration file 10-48
  predefined checks 7-282
  product main features 2-1
  protecting external Java resources on Node.js servers 7-275
  protecting external Java resources on WebSphere servers 7-275
  protecting resources 7-271
  resource protection
    adapters 7-272
    disabling 7-272
    external resources 7-274
  resource server 7-265
  Transport Layer Security v1.2 6-165
  web applications 7-73
security 7-265 *(continued)*
    Chrome secure-origins policy 7-73
    single-origin policy 7-73
  what's new in 8.0 3-10
security API 7-203
security checks 7-265, 7-281, 7-291
  base classes 7-289
  configuring 7-297, 7-298, 7-300
    predefined LTPA-based single sign-on (SSO) security check 7-289
  contract 7-291
  defining 7-289, 7-294
  defining scope mapping elements 7-263
  definition 7-295
  device single sign-on configuration 7-301
  implementing 7-289
  interface 7-289, 7-291
  predefined 7-282
    application authenticity 7-282
    LTPA-based single sign-on (SSO) security check 7-285
  properties 7-297
    expirationSec 7-289
security framework 7-265
security utilities 7-181, 7-182, 7-183, 7-184
self-signed certificates
  CA certificates, managing and installing in an Application Center test environment 6-260
  not supported on Application Center 6-257
  to configure SSL 10-3
sending 11-37, 11-38, 11-39, 11-40, 11-41, 11-42, 11-43
Server Configuration Tool
  Applying a fix pack 6-110
  creating database tables for MobileFirst Server 6-70
  creating DB2 database for MobileFirst Server 6-71
  creating MySQL database for MobileFirst Server 6-73
  creating Oracle database for MobileFirst Server 6-72
  install server farm 6-140
  installation of MobileFirst Server in graphical mode 6-5
  installing MobileFirst Server to an application server 6-105
  known limitation 3-25
  starting and running 6-106
  Supported operating systems
    Linux x86 or Linux x86-64 6-106
    Mac OS x86-64 6-106
    Windows x86 or x86-64 6-106
  Supported topologies 6-106
server farm
  reverse proxy 6-98
server farms
  configuring 6-144
  deploying MobileFirst Server to IBM PureApplication 9-44

server farms *(continued)*
  heartbeat rate, timeout values, and status 6-149
  homogeneous, as opposed to heterogeneous 6-139
  installing with Ant tasks 6-141
  installing with the Server Configuration Tool 6-140
  invalid configuration 6-139
  planning the configuration 6-139
  tutorial about the installation MobileFirst Server in command line mode 6-22
  tutorial about the installation MobileFirst Server in graphical mode 6-5
  verifying configuration 6-148
  when to declare 6-139
server topologies
  MobileFirst Server 6-78
server-side artifacts
  transferring the configuration to a test or production server 10-7
server-side development
  changes from V6.2 to V8.0.0 5-1
servers
  transferring application configurations 10-9
Service Level Agreement (SLA)
  and WebSphere Application Server Network Deployment topology 6-94
Setting
  application license 10-80
setting up your environment for developing native Android applications 7-53
setting up your environment for developing native iOS applications 7-27
setup 7-182
  MobileFirst Server databases 6-63
sharing
  applications through Application Center 13-1
shells
  not supported in V.8.0.0 5-1
sideloading Windows applications 13-2
silent installation
  command line parameters 6-51
  using XML response files 6-47
single sign-on (SSO)
  device
    configuring 7-301
  LTPA-based predefined security check 7-285
    configuring 7-289
Single Sign-On (SSO)
  configuring a server farm 6-144
skins
  not supported in V.8.0.0 5-1
SLA
  See Service Level Agreement (SLA)
SOAP-based service request 7-219
software development kits
  supported 2-7
splash screens
  as Cordova application resources 7-102
  custom 7-104
SQL adapter
  connectionPolicy 7-213
  elements 7-213
SQL adapter XML file 7-213
SQL adapter XML file structure 7-213
SQL adapters
  example 7-222
  procedures 7-222
SSL
  configuring between adapters and back-end servers 10-3
  configuring for Application Center 6-257
  JNDI properties
    for MobileFirst Server administration service 6-174
  keystore 7-316
  security with a server farm 6-139
  to connect to Application Center from an iOS login view 13-61
SSL/TLS 7-185
stars
  to rate appreciations of installed applications on Application Center 13-86
stylesheets
  for Cordova applications 7-102
support
  unsubscribing a device from SMS notification
    not supported in 8.0 6-166
supported platforms
  token licensing 6-159
SWAM (WebSphere authentication method)
  configuring user authentication for MobileFirst Server administration 6-166
symmetric deployment
  Liberty collective 6-91
  WebSphere Application Server Network Deployment 6-94
synchronization
  of device with application log profile 11-24
system messages 8-4
system requirements 11-2

T

tablets
  See also ARM-based tablets
  reviews of application versions 13-86
  showing details of a specific application version 13-79
  submitting reviews of installed applications 13-86
tag-based notifications
  sending to the device 7-258
tags
  for tag-based notification 7-257
TAI
  See Trust Association Interceptor (TAI)
template
  Cordova 7-88
test environments
  building applications 10-4
  managing and installing self-signed CA certificates 6-260
  transferring application configurations 10-7
test servers
  transferring server-side artifacts 10-7
testing adapter
  CLI 7-230
  command line interface 7-230
testing applications
  product main features 2-1
thumbnail images
  as Cordova application resources 7-102
timeout values
  for server farm nodes 6-149
Tivoli Endpoint Manager
  propert to manage mobile applications 13-27
TLS v1.2
  See Transport Layer Security v1.2
token licenses
  validation 10-83
token licensing 2-8
  blocked application status 10-20
  installation overview 6-151
  insufficient tokens and application status in MobileFirst Operations Console 10-20
  planning
    Development 6-150
    installation process 6-150
    Operations 6-150
    supported platforms 6-150
    supported topologies 6-150
    technical restrictions 6-150
  troubleshooting 6-160
tokens 2-8
Tomcat 6-170
  configuring a server farm 6-144
  configuring Derby manually for Application Center 6-214
  configuring for DB2 manually for Application Center 6-210
  configuring for LDAP authentication 6-246
  configuring LDAP authentication 6-246
  logging and monitoring mechanisms 6-194
  logging and tracing 13-88
  starting and running Server Configuration Tool 6-106
  troubleshooting a corrupted login page 13-20
  troubleshooting MobileFirst Operations Console calls 6-194
  troubleshooting token licenses 6-160
tools 7-13

topologies
  multiple instances of MobileFirst Server on the same server or WebSphere Application Server cell 6-100
  multiple MobileFirst runtimes 6-99
  server farm 6-88
  stand-alone 6-85
topology
  asymmetric deployment 6-91, 6-94
  Liberty collective 6-91
  reverse proxy with server farm and WebSphere Application Server Network Deployment 6-98
  WebSphere Application Server Network Deployment 6-94
topology constraints
  of MobileFirst runtime 6-84
  of MobileFirst Server administration service 6-84
  of MobileFirst Server live update service 6-84
traces
  See logging
tracing
  See also logging and tracing
  JNDI properties for controlling Application Center trace output 13-88
tracking licenses 10-80
translation 8-4
Transport Layer Security (TLS)
  enforcing secure connections in iOS apps 7-46, 7-127
Transport Layer Security v1.2
  configuring MobileFirst Server 6-165
troubleshooting 11-49, 14-1
  corrupted Application Center login page in Safari browsers 13-21
  licence validation failure 10-83
  logging and tracing for Application Center 13-87
  mfpadm Ant task commands 10-44
  mfpadm program commands 10-71
  MobileFirst Operations Console calls, with MySQL on Tomcat 8 6-194
  push notification on iOS 7-265
  token licensing 6-160
  Tomcat corrupted login page 13-20
Trust Association Interceptor
  MobileFirst OAuth TAI 7-274
Trust Association Interceptor (TAI)
  MobileFirst OAuth TAI 7-275
truststores
  and SSL configuration 6-197
tutorials 4-1
  installation of MobileFirst Server 6-4

U

Unicode Normalization Forms
  limitation on search 3-25
uninstallanalytics
  Ant task 6-309
uninstalling
  applications 13-78
uninstallmobilefirstadmin
  Ant task 6-275
uninstallmobilefirstpush
  Ant task 6-287
uninstallmobilefirstruntime
  Ant task 6-293
United States Government Configuration Baseline 10-74, 10-75
Universal Windows Platform (UWP) 7-65
unresponsive servers
  heartbeat mechanism 6-149
updateanalytics
  Ant task 6-309
updatemobilefirstadmin
  Ant task 6-275
updatemobilefirstpush
  Ant task 6-287
updatemobilefirstruntime
  Ant task 6-293
updates
  configuring push notification 13-13
updating client apps directly
  CLI 7-236
  development and production environments 7-235
  full vs. incremental 7-235
  keystore 7-235
  lifecycle 7-236
  MobileFirst Operations Console 7-236
  overview 7-235
  secure and non-secure 7-235
upgrade path 5-1
upgrades
  V6.2 applications to V8.0.0 5-1
upgrading
  native Android projects 5-25
  native iOS projects 5-17, 5-23, 5-33, 5-44, 5-45
  native Windows application 5-31
upgrading, native iOS 5-19
upload, Ant task 13-49
user authentication
  See authentication
user roles
  configuring authentication for MobileFirst Server administration 6-166
users 11-36
  mapped to Java EE security roles for Application Center 6-234, 6-236
users and groups
  access to Application Center 6-238, 6-240, 6-242
  Application Center 6-238

V

verbosity of client log files 11-24
verification
  server farm configuration 6-148
version 5-1
versions of applications
  in Application Center 13-2
  reverting 13-85
  reviewing 13-86
versions of applications *(continued)*
  showing details of a specific version 13-79
views
  in the Application Center client 13-63
Virtual Member Manager (VMM)
  Application Center Access Control List 6-238, 6-240
  JNDI properties for Application Center 6-261
Visual Studio
  application packages 13-22
  to build the Application Center mobile client 13-6
VMM
  See Virtual Member Manager (VMM)

W

watchOS 7-48, 7-51
web
  applications
    registering 7-80
    registering using MobileFirst Operations Console 7-81
    registering using MobileFirst Platform CLI 7-80
web applications
  analytics 7-73, 7-76, 7-79
  developing 7-1, 7-73
    acquiring the SDK 7-75
    initializing the SDK 7-79
    using the SDK 7-76
web browsers
  supported 2-7
web resources
  checksum
    configuring 7-20
    test ignored file extensions 7-20
  packaging and uploading to MobileFirst Server 7-237
WebSphere Application Server
  configuring for Application Center 6-224
    after EAR file deployment 6-231
  configuring for DB2 manually for Application Center 6-209
  configuring for Derby manually for Application Center 6-212
  configuring Java EE security roles for Application Center 6-234
  configuring LDAP ACL management for Application Center 6-240
  configuring LDAP ACL management for V7 6-238
  configuring profile for MobileFirst Server administration 6-169
  configuring the endpoint of application resources for Application Center 6-253
  configuring to avoid MySQL timeout issues 6-192
  configuring to avoid MySQL, DB2, and Oracle timeout issues 6-192

WebSphere Application Server *(continued)*
configuring user authentication for MobileFirst Server administration 6-166
deploying MobileFirst Server to IBM PureApplication 9-44
installing MobileFirst Analytics 11-7
logging and monitoring mechanisms 6-194
LTPA-based single sign-on (SSO) 7-285
managing ACL for Application Center with LDAP 6-240
manual configuration 6-220
multiple instances of MobileFirst Server 6-100
outbound dynamic configuration 6-197
prerequisites for installing MobileFirst Server 6-104
protecting external resources 7-275
resource protection 7-274
SSL configuration and HTTP adapters 6-197
starting and running Server Configuration Tool 6-106
V7.0 and V8.0 are no longer supported 5-1
WebSphere Application Server full profile
logging and tracing in Application Center 13-87
topologies 6-85, 6-88
WebSphere Application Server Liberty
configuring a server farm 6-144
configuring for Application Center manually
after EAR file deployment 6-229
configuring Java EE security roles for Application Center 6-236
configuring LDAP ACL management 6-243
configuring LDAP authentication 6-242
configuring profile for MobileFirst Server administration 6-169
configuring the endpoint of application resources for Application Center 6-255
deploying MobileFirst Server to IBM PureApplication 9-44
installation of MobileFirst Server in command line mode 6-22
installation of MobileFirst Server in graphical mode 6-5
installing MobileFirst Analytics 11-4
prerequisites for installing MobileFirst Server 6-103
resource protection 7-274
starting and running Server Configuration Tool 6-106
WebSphere Application Server Liberty collective
See also Liberty collective
installing MobileFirst Server components manually 6-121
starting and running Server Configuration Tool 6-106
WebSphere Application Server Liberty profile
Application Center installation 13-6
configuring to avoid MySQL, DB2, and Oracle timeout issues 6-192
logging and tracing in Application Center 13-88
topologies 6-85, 6-88
WebSphere Application Server Network Deployment
prerequisites for installing MobileFirst Server 6-104
reverse proxy 6-98
topologies 6-94
troubleshooting token licenses 6-160
WebSphere Application Server V8
configuring LDAP for Application Center 6-238
WebView interface
developing Cordova applications 7-83
what's new 3-1, 3-17
Android support 5-17
API 3-3
deploying and managing apps 3-10
discontinued features 3-19
external library 3-3
in building applications 3-1
interim fixes 3-16
migrating existing apps 5-17
new mobile OS updates 5-17
removed features 5-4
whitelisting
See Restricting access to the consoles running on containers
Windows
adding MobileFirst SDK to an application 7-66
adding MobileFirst SDK to an application manually 7-65
adding optional MobileFirst components to an application 7-67
applications
registering 7-68
registering using MobileFirst Operations Console 7-70
registering using MobileFirst Platform CLI 7-68
known limitations 3-25
push notifications 7-256
stopping the MobileFirst Development Server 7-13
Windows 10 Universal Windows Platform (UWP) 7-72
Windows 8
developing native applications 7-1, 7-24
file types 13-2
sideloading applications 13-2
specific platform requirements for Application Center 13-2
Windows 8 Universal 7-72, 7-170, 7-171
adding mobile applications to Application Center 13-22
developing native applications 7-65
framework packages 13-22
Windows 8 Universal *(continued)*
prerequisites for using the mobile client 13-6
updating applications in Application Center 13-80
Windows applications
getting started 7-9
Windows devices
showing details of a specific application version 13-79
views in the Application Center client 13-63
Windows Phone
deploying the mobile client in Application Center 13-12
installing and running the mobile client in Application Center 13-6
removing applications 13-78
Windows Phone 8
developing native applications 7-1, 7-24
Windows Phone applications
properties 13-27
reverting 13-85
Windows Phone devices
reviews of application versions 13-86
showing details of a specific application version 13-79
views in the Application Center client 13-63
Windows Phone Store
not supported for adding applications toApplication Center catalog 13-24
Windows Phones
preparation for the mobile client 13-8
Windows push notification server
WNS 7-256
Windows Store
not supported for adding applications toApplication Center catalog 13-24
removing applications 13-78
Windows Universal applications
migration scenarios for push notification 5-75
WindowsPhone 8
specific platform requirements for Application Center 13-2
WinJS framework package
adding Windows 8 mobile applications to Application Center 13-22
WKWebView
plug-in for iOS applications 7-134

X

X.509 certificates
See self-signed certificates
Xcode 7-48
XML response files
installing Application Center 6-47
XSL transformation filtering 7-219