Red Hat JBoss Fuse 6.2

Migration Guide

Migrating to Red Hat JBoss Fuse 6.2

Last Updated: 2017-09-27

Red Hat JBoss Fuse 6.2 Migration Guide

Migrating to Red Hat JBoss Fuse 6.2

JBoss A-MQ Docs Team
Content Services
fuse-docs-support@redhat.com

Legal Notice

Abstract

This guide lays out the issues a user will encounter when upgrading to the latest version of Red Hat JBoss Fuse.

Table of Contents

CHAPTER 1. MIGRATION OVERVIEW

Abstract

This chapter highlights some of the key points that might affect your applications, when migrating from JBoss Fuse 6.1 to JBoss Fuse 6.2. For a detailed description of the changes made to each of the components, see the relevant chapters for Apache ActiveMQ, Apache Camel, and Apache CXF.

1.1. UPGRADED COMPONENTS

Version upgrades

Many of the major components in JBoss Fuse and JBoss A-MQ 6.2 have been upgraded. The following versions are used in JBoss Fuse:

Table 1.1. Component Versions

Component	Version for 6.1	Version for 6.2
Apache ActiveMQ	5.9.0	5.11.0
Apache Camel	2.12.0	2.15.1
Apache CXF	2.7.0	3.0.4
Apache Karaf	2.3.x	2.4.0
Fabric8 (was Fuse Fabric)	1.0.0	1.2.0
Spring framework	3.2.x	3.2.x

XA transactions

If you have deployed any transaction code that relies on the Apache Aries auto-enlisting XA wrapper, you should note that this feature is no longer installed by default. To enable the Aries XA wrapper service, you must now install the **connector** features, as follows:

```
JBossFuse:karaf@root> features:install connector
```

Apache ActiveMQ changes

The changes resulting from the upgrade to version 5.11.0 are detailed in Chapter 5, *Apache ActiveMQ Issues*.

Apache Camel changes

The changes resulting from the upgrade to version 2.15.1 are detailed in Chapter 6, *Apache Camel Issues*.

The most important changes are:

- The Camel SAP component has been refactored into ten separate sub-components and now also supports sending and receiving IDoc documents. For details, see chapter "SAP Component" in "Apache Camel Component Reference".
- Camel supports a new REST DSL, which makes it easy to define REST services in a Camel route. For details, see chapter "Defining REST Services" in "Apache Camel Development Guide".

Apache CXF changes

The changes resulting from the upgrade to version 3.0.4 are detailed in Chapter 7, *Apache CXF Issues*.

The most important changes are:

- The `cxf-api` bundle and the `cxf-rt-core` bundle are no longer available in Apache CXF 3.0.x. For details, see Section 7.3, "Packaging of Bundles/JARs".
- The `http://cxf.apache.org/jaxrs` namespace has been changed to `http://cxf.apache.org/jaxrs-client` in Apache CXF 3.0.x. For details, see Section 7.4, "XML Schema Namespaces".

1.2. NEW ROLE-BASED ACCESS CONTROL

Overview

JBoss Fuse 6.2 has a new role-based access control (RBAC) feature, which is enabled by default in the container. The new RBAC system offers differentiated access to the container, depending on which roles have been assigned to a user. The RBAC imposes access controls on the Karaf console (so that only administrators can access the full range of commands and command options) and imposes access controls on the JMX protocol (so that access control is applied to the Fuse Management Console).

Standardized roles

Table 1.2, "Standard Roles for Access Control" lists and describes the standard roles that are used throughout the JMX ACLs and the command console ACLs.

Table 1.2. Standard Roles for Access Control

Roles	Description
`Monitor`, `Operator`, `Maintainer`	Grants read-only access to the container.
`Deployer`, `Auditor`	Grants read-write access at the appropriate level for ordinary users, who want to deploy and run applications. But blocks access to sensitive container configuration settings.
`Administrator`, `SuperUser`	Grants unrestricted access to the container.

Migrating user data for RBAC

When migrating to JBoss Fuse 6.2, you must modify your user data, so that users are assigned one of the standard roles from Table 1.2, “Standard Roles for Access Control”.

Reference

For more details about role-based access control, see section "Role-Based Access Control" in "Security Guide".

1.3. SECURITY CHANGES

Overriding the default JAAS realm in Fabric

In JBoss Fuse 6.2, the rank of the default **`ZookeeperLoginModule`** JAAS module (which is installed by default in a Fabric container) has changed to **`99`**, and the name of the default realm is **`karaf`**. In previous releases, the rank of **`ZookeeperLoginModule`** realm was just **`1`**.

Hence, if you want to override the default **`karaf`** in the context of Fabric, you must define a new realm named **`karaf`**, with a **`rank`** attribute that is greater than or equal to **`100`**.

Enabling LDAP authentication in a Fabric

In particular, this change affects the configuration needed to enable LDAP authentication in a Fabric. In this case, the **`rank`** attribute of the **`jaas:config`** element in the JAAS realm configuration file must be increased to at least **`100`** (recommended is **`200`**). For details, see section "Procedure for a Fabric" in "Security Guide".

1.4. MIGRATING MAVEN PROJECTS

Overview

JBoss Fuse 6.1 now has a JBoss Fuse BOM (Bill of Materials), which defines the versions of all the JBoss Fuse Maven artifacts. You can use the BOM to simplify migration of your Maven POM files. Instead of updating the **`version`** elements on each Maven dependency, all you need to do is to import the latest JBoss Fuse BOM, which defines default versions for all of the dependencies provided by JBoss Fuse.

JBoss Fuse BOM

The JBoss Fuse BOM is a parent POM that defines the versions for all of the Maven artifacts provided by JBoss Fuse. The JBoss Fuse BOM exploits Maven's *dependency management* mechanism to specify default versions for the Maven artifacts, so that it is no longer necessary to specify the artifact versions explicitly in your POM.

Current version of the JBoss Fuse BOM

The easiest way to discover the current version of the JBoss Fuse BOM is to examine one of the sample **`pom.xml`** files from the **`quickstarts`** examples. For example, in the ***`InstallDir`*`/quickstarts/eip/pom.xml`** file, you can find a line that defines the JBoss Fuse BOM version, as follows:

```
<project ...>
    ...
    <properties>
```

```
        ...
        <!-- the version of the JBoss Fuse BOM, defining all the
dependency versions -->
        <jboss.fuse.bom.version>6.2.0.redhat-133</jboss.fuse.bom.version>
    </properties>
    ...
</project>
```

How to migrate Maven dependencies using the JBoss Fuse BOM

To migrate Maven dependencies using the JBoss Fuse BOM, open the Maven **pom.xml** file for your project and edit it as follows:

1. Define the JBoss Fuse BOM version as a property in your **pom.xml** file, using the current BOM version. For example:

```
<project ...>
    ...
    <properties>
        ...
        <jboss.fuse.bom.version>6.2.0.redhat-
133</jboss.fuse.bom.version>
    </properties>
    ...
</project>
```

2. Reference the JBoss Fuse BOM parent POM in a **dependencyManagement** element, so that it defines default versions for the artifacts provided by JBoss Fuse. Add the following **dependencyManagement** element to your **pom.xml** file:

```
<project ...>
    ...
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jboss.fuse.bom</groupId>
                <artifactId>jboss-fuse-parent</artifactId>
                <version>${jboss.fuse.bom.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    ...
</project>
```

3. Now delete all of the **version** elements in your JBoss Fuse dependencies. All of the versions defined in the JBoss Fuse BOM will be applied automatically to your dependencies (through the Maven dependency management mechanism). For example, if you already had some Apache Camel dependencies, as follows:

```
  <dependencies>
    <dependency>
      <groupId>org.apache.camel</groupId>
```

```
      <artifactId>camel-core</artifactId>
<version>2.15.1.redhat-620133</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-blueprint</artifactId>
<version>2.15.1.redhat-620133</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-jetty</artifactId>
<version>2.15.1.redhat-620133</version>
    </dependency>
    ...
 </dependencies>
```

You would delete the version elements, so that the dependencies are specified as follows:

```
 <dependencies>
   <dependency>
     <groupId>org.apache.camel</groupId>
     <artifactId>camel-core</artifactId>
   </dependency>
   <dependency>
     <groupId>org.apache.camel</groupId>
     <artifactId>camel-blueprint</artifactId>
   </dependency>
   <dependency>
     <groupId>org.apache.camel</groupId>
     <artifactId>camel-jetty</artifactId>
   </dependency>
   ...
 </dependencies>
```

4. In future, when you migrate to a later version of JBoss Fuse, all that you need to do to upgrade your **pom.xml** file is to edit the **jboss.fuse.bom.version** property, so that it references the new version of the JBoss Fuse BOM.

CHAPTER 2. UPGRADING TO A PRE-PATCHED DISTRIBUTION

WHAT IS A PRE-PATCHED DISTRIBUTION?

For version 6.2.0 of JBoss Fuse, pre-patched distributions are made available for the first time. A pre-patched distribution is an installation of the 6.2.0 GA version which has had the specified patch applied to it. This pre-patched installation is then re-packaged as an archive and made available for download.

For example, the following pre-patched distribution, which incorporates patch P1, is currently available for download:

```
Red Hat JBoss Fuse Pre-patched 6.2.0 P1
```

INSTALLING A PRE-PATCHED DISTRIBUTION

To install a pre-patched distribution, extract the downloaded archive file into a directory on your file system. The distribution is now ready for use as a standalone container (but *not* as a Fabric container—see the notes and limitations).

IMPORTANT NOTES AND LIMITATIONS

The pre-patched distribution is *not* equivalent to a complete rebuild of the product. This has some important implications, as follows:

- A pre-patched distribution is intended primarily to be used as a standalone container (that is, non-Fabric).
- If you do create a fabric using the pre-patched distribution (for example, using **`fabric:create`**), the Fabric container reverts to the unpatched, GA version of the product.
- To upgrade the Fabric to the relevant patch level, you must download the required patch (or paches) and follow the standard Fabric patching procedure, as described in .
- From version 6.2.1 of JBoss Fuse, the patching process will be improved so that pre-patched distributions can also be used to create properly patched Fabric containers.

CHAPTER 3. DEPRECATED AND REMOVED FEATURES

SERVICEMIX MAVEN ARCHETYPES NOT SUPPORTED

The ServiceMix Maven archetypes (with a **groupId** of **org.apache.servicemix.tooling**) are no longer supported and are not available in 6.2. You can use the fabric8 Maven archetypes instead (which provide similar functionality). The fabric8 archetypes have a **groupId** of **io.fabric8.archetypes** and the following fabric8 archetypes are available:

```
karaf-camel-amq-archetype
karaf-camel-cbr-archetype
karaf-camel-cxf-code-first-archetype
karaf-camel-cxf-contract-first-archetype
karaf-camel-dozer-wiki-archetype
karaf-camel-drools-archetype
karaf-camel-eips-archetype
karaf-camel-errorhandler-archetype
karaf-camel-log-archetype
karaf-camel-log-wiki-archetype
karaf-camel-webservice-archetype
karaf-rest-archetype
karaf-secure-rest-archetype
karaf-secure-soap-archetype
karaf-soap-archetype
```

FUSE APPLICATION BUNDLES

Fuse Application Bundles (FABs) are no longer supported and are not available in 6.2. Instead of using FABs, it is recommended that you repackage your code as an OSGi bundle, for deployment into the JBoss Fuse container.

JBI CONTAINER

The Java Business Integration (JBI) container has been removed from JBoss Fuse 6.2.

APACHE OPENJPA IS DEPRECATED

The Apache OpenJPA implementation of the Java Persistence API (JPA) is deprecated in 6.2. It is recommended that you use the Hibernate implementation instead.

SPRING DYNAMIC MODULES (SPRING-DM) IS DEPRECATED

Spring-DM (which integrates Spring XML with the OSGi service layer) is deprecated in 6.2 and you should use the Blueprint framework instead. Using Blueprint does not prevent you from using the Spring framework: the latest version of Spring is compatible with Blueprint.

THE CAMEL-INFINISPAN COMPONENT HAS BEEN REMOVED FROM JBOSS FUSE 6.2

The **camel-infinispan** component is not included in JBoss Fuse 6.2. It is recommended that you use the **camel-jbossdatagrid** component instead. For details on how to install it, see *Red Hat JBoss Data Grid and Red Hat JBoss Fuse* in the Red Hat JBoss Data Grid *Getting Started* guide.

You can use any supported combination of JBoss Data Grid in a given container, where the list of certified versions is listed in the JBoss Data Grid support matrix. For some examples of how to use JBoss Data Grid with JBoss Fuse, see https://github.com/jbossdemocentral/jdg-fuse-demos.

CHAPTER 4. CONSOLE CHANGES

FABRIC:MQ-CREATE COMMAND

The following argument names have changed in JBoss Fuse 6.2:

- **`--ports`** to **`--port`**
- **`--networks`** to **`--network`**

FABRIC:PROFILE-CREATE COMMAND

The following argument names have changed in JBoss Fuse 6.2:

- **`--parents`** to **`--parent`**

FABRIC:PROFILE-EDIT COMMAND

The following argument names have changed in JBoss Fuse 6.2:

- **`--repositories`** to **`--repository`**
- **`--features`** to **`--feature`**
- **`--libs`** to **`--lib`**
- **`--bundles`** to **`--bundle`**

FABRIC:EXPORT AND FABRIC:IMPORT COMMANDS

The **`fabric:export`** and **`fabric:import`** commands have been removed in JBoss Fuse 6.2, and are now replaced by the corresponding **`zk:export`** and **`zk:import`** commands. To gain access to these Zookeeper commands, you must install the **`fabric-zookeeper-commands`** feature.

NOTE

The **`zk:export`** and **`zk:import`** commands interact purely with the Zookeeper registry. For example, you cannot use these commands to export or import Fabric profile data, which is now stored in the container's Git repository.

CHAPTER 5. APACHE ACTIVEMQ ISSUES

Abstract

JBoss Fuse 6.2 uses Apache ActiveMQ 5.11.0. Since the last release, Apache ActiveMQ has been upgraded from version 5.9.0 to version 5.11.0. This introduces a few migration issues.

5.1. MIGRATING CLIENTS

Migrating Apache ActiveMQ clients

In general, it is recommended that you update your Apache ActiveMQ clients at the same time that you update the brokers, in order to guarantee compatibility between clients and brokers.

It is possible, in some cases, that older client versions might be interoperable with later broker versions. The Openwire protocol supports version negotiation, such that an old client can negotiate the lowest common version with its peer and use that version. But JBoss Fuse does not have a comprehensive test suite for testing compatibility between all of the different versions of Apache ActiveMQ. Hence, to be sure of compatibility, it is recommended that you upgrade your clients along with your brokers to use the same version.

5.2. NEW FEATURES

ActiveMQ 5.10.0

In ActiveMQ 5.10.0, the following new features have been introduced:

- Hardened MQTT support
- Hardened AMQP support
- Hardened LevelDB store
- Improved RAR/JCA adapter
- Improved Runtime configuration plugin

ActiveMQ 5.11.0

In ActiveMQ 5.11.0, the following new features have been introduced:

- Destination import/export for lock down mode. Use the **`destinationsPlugin`** on the broker to import/export broker destinations to s a specified location. For example:

```
<plugins>
    <destinationsPlugin location="/workspace/destinations"/>
</plugins>
```

- Dynamic Camel root loading. Allows routes to modified on the fly without restarting the broker. To take advantage of this feature, you must define a **`camelroutesBrokerPlugin`** plug-in in the broker configuration, as follows:

```
<plugins>
    <camelroutesBrokerPlugin routesFile="routes.xml" />
</plugins>
```

Where the **routes.xml** file must be in the same location as the broker configuration file.

- MQTT: QOS2 mapped to virtual topics. Can be enabled using the transport option, **?transport.subscriptionStrategy="mqtt-virtual-topic-subscriptions"**.
- Start scripts simplification
- Recover scheduler database option

5.3. DEPENDENCY UPGRADES

Spring framework

JBoss Fuse and JBoss Fuse use Spring framework version 3.2.

Apache Karaf

JBoss Fuse and JBoss Fuse use Apache Karaf version 2.4.0.

5.4. API CHANGES

JMS streams

JMS streams are now *deprecated*. If you need to send very large messages, it is recommended that you use an out-of-bounds transport, such as FTP, instead. In particular, the **org.apache.activemq.ActiveMQInputStream**and **ActiveMQOutputStream** classes are deprecated, as are the **ActiveMQConnection.createInputStream** and **ActiveMQConnection.createOutputStream** methods.

Camel ActiveMQ component

Removed the **org.apache.activemq.camel.converter.IdentityMessageReuseConverter** class from the Camel ActiveMQ component (**activemq-camel**).

CHAPTER 6. APACHE CAMEL ISSUES

6.1. SUMMARY OF APACHE CAMEL 2.12 TO APACHE CAMEL 2.15.1 MIGRATION

Overview

Red Hat JBoss Fuse 6.2.0.redhat-133 uses Apache Camel 2.15.1. Since the last release, Apache Camel has been upgraded from version 2.12 to version 2.15.1. This introduces a few migration issues.

Spring framework

Spring 3.0.x and Spring 3.1.x are no longer supported. You must use Spring 3.2.x.

The camel-infinispan component has been removed from JBoss Fuse 6.2

The **`camel-infinispan`** component is not included in JBoss Fuse 6.2. It is recommended that you use the **`camel-jbossdatagrid`** component instead. For details on how to install it, see *Red Hat JBoss Data Grid and Red Hat JBoss Fuse* in the Red Hat JBoss Data Grid *Getting Started* guide.

You can use any supported combination of JBoss Data Grid in a given container, where the list of certified versions is listed in the JBoss Data Grid support matrix. For some examples of how to use JBoss Data Grid with JBoss Fuse, see https://github.com/jbossdemocentral/jdg-fuse-demos.

New Camel components

The following new Camel components have been added in Apache Camel version 2.13:

HDFS2 component

Integration with HDFS using Hadoop 2.x client

JGroups component

provides exchange of messages between Camel infrastructure and JGroups clusters.

Apache Kafka component

Integration with Apache Kafka

OptaPlanner component

To use OptaPlanner for problem solving plans.

Splunk component

Enables you to publish and search for events in Splunk.

Amazon SWF component

For managing workflows running on Amazon's Simple Workflow Service.

The following new Camel components have been added in Apache Camel version 2.14:

AHC-WS component

Provides Websocket based endpoints for a client communicating with external servers over Websocket.

Atmosphere Websocket component

Provides Websocket based endpoints for a servlet communicating with external clients over Websocket (as a servlet accepting websocket connections from external clients).

Box component

Provides access to all of the Box.com APIs accessible using box-java-sdk-v2.

Dropbox component

Enables you to treat Dropbox remote folders as a producer or consumer of messages.

Metrics component

Enables you to collect various metrics directly from Camel routes (using Metrics).

Netty4 component

A socket communication component, based on the Netty project version 4.

Netty4-HTTP component

An extension to Netty4 component to facilitate HTTP transport.

Olingo2 component

Uses Apache Olingo version 2.0 APIs to interact with OData 2.0 and 3.0 compliant services.

Openshift component

A component for managing your OpenShift applications.

Google Drive component

Provides access to the Google Drive file storage service via the Google Drive Web APIs.

Gora component

Enables you to work with NoSQL databases using the Apache Gora framework.

REST component

Enables you to define REST endpoints using the Rest DSL and plug-in to other Camel components as the REST transport.

Spark-REST component

Enables you to define REST endpoints using the Spark REST Java library using the Rest DSL.

Schematron component

An XML-based language for validating XML instance documents.

Swagger component

Enables users to create API docs for any REST-defined routes and endpoints in a CamelContext file.

The following new Camel components have been added in Apache Camel version 2.15:

Beanstalk component

For working with Amazon Beanstalk jobs.

Cassandra component

Cassandra CQL3 support.

Chunk component

For templating with Chunk engine.

Docker component

To communicate with Docker.

Dozer component

Now also as a component to convert messages using the Dozer type conversion framework.

GitHub component

For integrating with github.

Google Calendar component

Provides access to Google Calendar through the Google Calendar Web APIs.

Google Mail component

Provides access to Gmail via the Google Mail Web APIs.

Hipchat component

To integrate with the Hipchat service.

PGEvent component

For sending/receiving notifications in PostgreSQL via the pgjdbc-ng driver.

JIRA component

For integrating with JIRA issue tracker.

Kura component

For deploying Camel OSGi routes into Eclipse Kura M2M container.

SCR component

For using Camel with SCR (OSGi declarative services) on OSGi containers such as Apache Karaf.

Spring Boot component

For using Camel with Spring Boot.

Test-Spring40 component

For testing with Spring 4.0.x. `camel-test-spring` is for Spring 4.1.x onwards.

Scheduler component

For timer based scheduler using a scheduled thread pool and with more functionality.

New languages

The following new expression languages have been added in Apache Camel version 2.13:

JSonPath

A language syntax (similar to XPath) for extracting parts of JSon messages.

The following new expression languages have been added in Apache Camel version 2.14:

XMLTokenizer

A truly XML-aware tokenizer that can be used with the Splitter as the conventional Tokenizer to efficiently and effectively tokenize XML documents.

New DSL

The following new DSL commands have been added in Apache Camel version 2.15:

`removeProperties`

A DSL command to remove exchange properties.

New Maven archetypes

The following new Apache Camel Maven archetypes have been added in Apache Camel versions 2.13, 2.14, and 2.15:

- **`camel-archetype-cxf-code-first-blueprint`**
- **`camel-archetype-cxf-contract-first-blueprint`**

6.2. SPRING FRAMEWORK

Spring version

Since Apache Camel 2.13, the version of Spring that you can use with Apache Camel *must* be 3.2 (or later). Spring 3.0.x and 3.1.x are not supported.

New schema location

If you explicitly specify the location of the Spring schema in your Spring configuration files, you must change the schema location to point at the 3.2 Spring schema.

The Spring 3.2 schema is located at the following Web page:

```
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
```

For example, assuming your schema locations are specified in the root **beans** element, you could specify the new Spring schema location as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-
3.2.xsd">
```

Spring 3.2 new features

For a summary of the new features in Spring 3.2, see New Features and Enhancements in Spring 3.2 .

Using the Spring registry in Apache Camel

Since Apache Camel 2.11, when using Spring's bean registry with Camel, Camel also looks up Spring's ancestor application contexts.

6.3. PRODUCT DEPENDENCIES

Apache CXF version

Since Apache Camel 2.12, the CXF component requires Apache CXF 2.7.x.

Jetty version

Since Apache Camel 2.13, the Jetty component is updated to Jetty 8.1.12.

Component dependencies

In Apache Camel 2.13, 2.14, and 2.15 some components have had their third party dependencies upgraded, as follows:

- ActiveMQ 5.8.0 to 5.11.0
- AHC 1.7.20 to 1.8.3
- APNS 0.1.6 to 0.2.3
- Atomikos 3.8.0 to 3.9.3
- AWS-SDK 1.5.1 to 1.8.9.1
- Avro 1.7.3 to 1.7.5
- BeanIO 2.0.6 to 2.0.7
- Classmate from 0.8.0 to 1.0.0
- Codehale Metrics 3.0 to 3.1
- Commons Codec 1.8 to 1.9

- Commons Httpclient 4.2.5 to 4.3.3
- Commons Httpcore 4.2.4 to 4.3.2
- CXF 2.7.6 to 3.0.4
- Deltaspike 0.7 to 1.0.1
- Disruptor 3.3.0 to 3.3.2
- Dozer 5.4.0 to 5.5.1
- Ehcache 2.7.2 to 2.8.3
- Elasticsearch 0.20.6 to 1.0.0
- FOP 1.0 to 1.1
- Groovy 2.2.2 to 2.3.4
- Guava 14.0.1 to 18.0
- Hadoop 1.2.0 to 1.2.1. Hadoop 2.3.0 supported by camel-hdfs2 component.
- Hapi 2.1 to 2.2
- Hazelcast 2.6 to 3.4
- Hibernate Validator 5.0.1.Final to 5.0.3.Final
- ICal4j 1.0.4 to 1.0.5.2
- Jackson 2.2.2 to 2.3.2
- JAX-B 2.2.7 to 2.2.11
- Jclouds 1.6.2-incubating to 1.7.0
- Jettison 1.3.4 to 1.3.5
- Jetty 7.6.9 to 8.1.12
- JLine 0.9.94 to 2.11
- Joda time 2.1 to 2.3
- JRuby 1.7.4 to 1.7.18
- JSCH 0.1.49 to 0.1.51
- JsonPath 1.1.0 to 1.2.0
- LevelDb JNI 1.7 to 1.8.
- Lucene 3.6.0 to 4.6.1
- MongoDB Java Driver 2.11.2 to 2.12.0

- Mustache 0.8.12 to 0.8.13
- MVEL 2.1.6.Final to 2.1.7.Final
- MyBatis 3.2.2 to 3.2.5
- Netty3 3.8.0.Final to 3.9.0.Final
- OGNL 3.0.6 to 3.0.8
- Pax Logging 1.6.10 to 1.7.1
- Protobuf 2.3 to 2.5
- Qpid 0.20 to 0.26
- Quartz 2.2.0 to 2.2.1
- RabbitMQ amqp Java Client 3.1.3 to 3.3.0
- Restlet 2.0.15 to 2.2.1
- RxJava 0.11.1 to 1.0.5
- Saxon 9.5.0.2 to 9.5.1-4
- Scala 2.10.2 to 2.11.2
- Servlet API 2.5 to 3.0
- Shiro to 1.2.3.
- Slf4j 1.7.5 to 1.7.6
- Snappy 1.0.4.1 to 1.1.0.1
- SNMP4J 2.2.2 to 2.3.0
- SolrJ 3.6.2 to 4.6.1
- Spring Batch 2.2.1.RELEASE to 2.2.2.RELEASE
- Spring Integration 2.2.4.RELEASE to 2.2.6.RELEASE
- Spring Redis 1.0.4.RELEASE to 1.3.4.RELEASE
- Spring Security 3.1.7.RELEASE to 3.2.5.RELEASE
- Spring WS 2.1.3.RELEASE to 2.1.4.RELEASE
- SSHD 0.8.0 to 0.11.0
- StompJMS 1.17 to 1.19
- TestNG 6.8.5 to 6.8.7
- Twitter4j 3.0.3 to 4.0.1

- Weld 1.1.5.Final to 1.1.18.Final
- XBean Spring 3.14 to 3.16
- XmlSec 1.5.5 to 1.5.6
- XStream 1.4.4 to 1.4.7

6.4. API CHANGES

API changes

The following changes have been made to the Java API:

CamelContext

Since Apache Camel 2.13, added **`getRegistry(T)`** method to **`org.apache.camel.CamelContext`** class.

Since Apache Camel 2.14, added **`startAllRoutes`** method to **`org.apache.camel.CamelContext`** class.

HttpClientConfigurer

Since Apache Camel 2.13, the signature of the **`configureHttpClient(HttpClient client)`** method from the **`org.apache.camel.component.http4.HttpClientConfigurer`** interface has been changed to **`configureHttpClient(HttpClientBuilder clientBuilder)`**.

ManagedCamelContextMBean

Since Apache Camel 2.14, the duplicate **`getMessageHistory`** method has been removed from the **`org.apache.camel.api.management.mbean.ManagedCamelContextMBean`** interface. Use the **`isMessageHistory`** method instead.

SynchronizationRouteAware

Since Apache Camel 2.14, added the **`org.apache.camel.spi.SynchronizationRouteAware`** class.

UnitOfWork

Since Apache Camel 2.14, added the **`beforeRoute`** and the **`afterRoute`** methods to the **`org.apache.camel.spi.UnitOfWork`** interface.

Moved classes

Since Apache Camel version 2.14, the following classes have moved to a different package or renamed:

- Renamed **`org.apache.camel.component.syslog.Rfc3164SyslogDataFormat`** to **`org.apache.camel.component.syslog.SyslogDataFormat`**

6.5. COMPONENT UPDATES

Jetty component

Since Apache Camel version 2.13.0, upgraded from 7.6.x to 8.1.x.

Hazelcast component

Since Apache Camel version 2.13.0, the Hazelcast component uses **`Object`** instead of **`String`** as the key type.

Since Apache Camel version 2.13.0, the atomic number producer uses a different atomic number name. This is due to the bug fix, CAMEL-6833 (previously, the wrong atomic number name was used).

Since Apache Camel version 2.13.0, the instance consumer writes headers to the *in* message (previously, the instance consumer was incorrectly writing headers to the *out* message instead of the *in* message).

Since Apache Camel version 2.13.0, the erroneous header value, **`envict`**, sent by the map/multimap consumer has now been corrected to **`evicted`**.

APNS component

Since Apache Camel version 2.13.0, the **`NON_BLOCKING enum`** value has been removed from the APNS component, because it is no longer support in APNS itself.

Language component

Since Apache Camel version 2.13.0, the Language component no longer caches the compiled script, because that could cause unwanted side-effects. You can set **`cacheScript=true`** to enable the previous behaviour, if you are sure that your script does not cause such side-effects.

Netty HTTP

Since Apache Camel version 2.13.0, the default value of the **`urlDecodeHeaders`** option on the Netty HTTP component has been changed from **`true`** to **`false`**.

Since Apache Camel version 2.14, Netty HTTP now removes the **`headerFilterStrategy`** option after resolving it.

Cache component

Since Apache Camel version 2.13.0, the Cache component no longer includes the **`ehcache.xml`** configuration file. The default EHCache configuration is used instead, if you do not specify a configuration file explicitly.

CDI component

Since Apache Camel version 2.13.0, the CDI component has been upgraded from DeltaSpike 0.3 to 0.5, which may affect upgrades.

SJMS component

Since Apache Camel version 2.13.0, when creating a consumer, the default value of **`NoLocal`** has changed from **`true`** to **`false`**.

ServletListener component

Since Apache Camel version 2.13.0, added **`beforeAddRoutes`** and **`afterAddRoutes`** methods to **`org.apache.camel.component.servletlistener.CamelContextLifecycle`** class.

Twitter component

Since Apache Camel version 2.14, the **`useSsl`** option has been removed, because SSL/TLS is now always enabled (this is enforced by Twitter).

Restlet component

Since Apache Camel version 2.14, the Restlet component has been fixed to return error code 405 (instead of 404), if a HTTP method is not allowed.

JMS and ActiveMQ components

Since Apache Camel version 2.14, routes starting with a JMS endpoint will now shut down the consumer (implemented by **`MessageListenerContainer`**) quicker when **`CamelContext`** is being stopped. This can help to achieve a cleaner shutdown, as otherwise some JMS clients might attempt to re-connect or fail over during the shutdown . If you want to disable this behaviour, so that the routes keep accepting messages during Camel shutdown, set **`acceptMessagesWhileStopping=true`**.

Bean component

Since Apache Camel version 2.14, this component caches the bean by default (that is, a single instance of the bean is used and re-used). This behaviour can be turned off by setting **`cache=false`**, which implies that a new instance of the bean is created for every method invocation (that is, there is no re-use).

XSLT component

Since Apache Camel version 2.15, the **`transformerFactory`** option now uses the with the ***`#BeanID`*** syntax to reference a bean instance.

Metrics component

Since Apache Camel version 2.15, Java endpoint API has been refactored somewhat.

CSV component

Since Apache Camel version 2.15, upgraded to Apache Commons CSV version 1.x, which has a different API to the 0.x version.

SJMS component

Since Apache Camel version 2.15, the API has been refactored.

Swagger component

Since Apache Camel version 2.15, the need for runtime-specific servlets in Swagger has been eliminated. You can just use the default servlet, which is provided out of the box.

Mail component

Since Apache Camel version 2.15, no longer sends headers whose key starts with **`Camel`**, as those are consider internal headers and should not be included in the sent emails.

SNMP component

Since Apache Camel version 2.15, the **`delay`** option has changed from using seconds to milliseconds as the time unit.

Bean component

Since Apache Camel version 2.15, you cannot use the Bean component as a consumer endpoint (**`from`**, at the start of a route). To get the same effect, you can start the route with a scheduler endpoint and send to a bean instance (using **`to("bean:`*`BeanID`*`")`** or **`beanRef("`*`BeanID`*`")`**).

6.6. MISCELLANEOUS CHANGES

DefaultTimeoutMap class

Since Apache Camel version 2.13.0, you must call the start method on the **`org.apache.camel.support.DefaultTimeoutMap<K,V>`** class to initialize the map before use.

@ExcludeRoutes annotation

Since Apache Camel version 2.13.0, the **`@ExcludeRoutes`** test annotation now accepts only classes implementing the **`RoutesBuilder`** interface.

MBean naming

Since Apache Camel version 2.13.0, the MBean names registered by Camel JMX no longer include the hostname in the context part. For example, in previous versions the context name would have the format, **`context=`*`myHost/myCamelId`***, whereas now it has the format, **`context=`*`myCamelId`***. Having the hostname in the MBean name makes things more complicated, because the MBean name changes depending on the host. The option, **`includeHostName`**, can be set to **`true`** in order to restore the old behaviour.

Since Apache Camel version 2.13.0, MBean naming in OSGi has been cleaned up to use simpler naming, with the MBean name now set to the OSGi **`symbolicName`** by default. In previous versions, it was possible for MBean names to have repeated bundle IDs—for example, **`context=114-114-camel-6`**.

jmxAgent element

Since Apache Camel version 2.13.0, the default value of the **`jmxAgent`** element's **`createConnector`** attribute has changed from **`true`** to **`false`**.

Since Apache Camel version 2.13.0, JMX Load statistics have been disabled by default. You can now explicitly enable statistics by setting **`loadStatisticsEnabled=true`** on the **`jmxAgent`** element.

onException Java DSL command

Since Apache Camel version 2.13.0, when using the **`onException.backOffMultiplier`** or **`collisionAvoidancePercent`** or **`collisionAvoidanceFactor`** options, back-off or collision avoidance is automatically enabled. In previous versions, it was also necessary to call

useExponentialBackOff or **useCollisionAvoidance** as well. The **errorHandler** DSL command has the same behaviour.

Spring XML shutting down beans

Since Apache Camel version 2.13.0, the **camelContext** bean will, by default, shut down before all other beans in Spring XML. This ensures a cleaner shutdown of the Camel context. If you prefer to revert to the old behaviour, you can set the new **camelContext** attribute, **shutdownEager**, to **false**.

onCompletion DSL command

Since Apache Camel version 2.14, now runs without a thread pool by default. To restore the old behaviour, set the **paralllelProcessing** option to **true**.

startAllRoutes method

Since Apache Camel version 2.14, setting **autoStartup** to **false**, and starting a **CamelContext** for the second time does *not* start the routes. Instead use the new **startAllRoutes** method on **CamelContext** to start all the routes.

Unit testing with Spring

Since Apache Camel version 2.15, unit testing with Spring 3.x now requires the **camel-test-spring3** component. The **camel-test-spring40** component is for testing with Spring 4.0.x and the **camel-test-spring** component is for testing with Spring 4.1.

Jetty 7.x

Since Apache Camel version 2.15, Jetty 7 has been dropped and is no longer supported.

Simple language and properties syntax

Since Apache Camel version 2.15, the property placeholder syntax has changed. It is now possible to specify a default value, using the syntax **${properties:*Key*:*DefaultVal*}**. Also, the syntax for overriding the properties file location has been changed from **${properties:*Location*:*Key*}** to **${properties-location:*Location*:*Key*}** (to avoid clashing with the new syntax for specifying default values).

Karaf camel:backlog-tracer-* commands

Since Apache Camel version 2.15, the Karaf console commands for the Camel backlog tracer (**backlog-tracer-dump**, **backlog-tracer-info**, **backlog-tracer-start**, **backlog-tracer-stop**) have been removed, because they are not suitable for a command-line environment.

InflightRepository interface

Since Apache Camel version 2.15, the **org.apache.camel.spi.InflightRepository** interface now includes additional methods for browsing in-flight exchanges.

exchangePattern URI option

Since Apache Camel version 2.15, when using the **`to`** DSL command or the **`recipientList`** DSL command, the exchange pattern specified by adding **`?exchangePattern=`*`Pattern`*** to the endpoint URI now takes precedence as the pattern that is used for invoking the endpoint.

Default class loader in CamelContext

Since Apache Camel version 2.15, the class loader, **`DefaultClassLoader`**, now falls back to use the application context classloader that may have been set on **`CamelContext`** to better be able to load classes/resources from the classpath in different runtime environments.

Removing a route

Since Apache Camel version 2.15, removing a route now also remove its static endpoints from the **`EndpointRegistry`** (if those endpoints are not shared and used by other routes). Note, however, that any dynamic endpoints created as a result of using dynamic Enterprise Integration Patterns such as recipient list, routing slip, dynamic router or similar, are not removed from the **`EndpointRegistry`** when the route is removed.

Model classes

Since Apache Camel version 2.15, all boolean **`is`*`Property`*** methods on the model classes have been removed, in order to ensure the model has consistent Java bean getter/setter style, with exactly one getter and one setter of each type.

Stream caching

Since Apache Camel version 2.15, added the **`copy`** method to **`StreamCache`** API for stream caching

@UriEndpoint annotation

Since Apache Camel version 2.15, if you use the **`@UriEndpoint`** annotation when implementing a custom Camel component, you must now also specify the **`syntax`** attribute to document the URI syntax of the endpoint.

ScalaRouteBuilder class

Since Apache Camel version 2.15, Scala developers should use the **`ScalaRouteBuilder`** class instead of the **`RouteBuilder`** class.

Exchange property language

The exchange property language has been renamed from **`property`** to **`exchangeProperty`**, in order to avoid ambiguity and confusion with properties as a general term.

CHAPTER 7. APACHE CXF ISSUES

7.1. SUMMARY OF APACHE CXF 2.7.0 TO 3.0.X MIGRATION

Overview

JBoss Fuse uses Apache CXF 3.0.x. Since the last release, Apache CXF has been upgraded from version 2.7.0 to version 3.0.x. This introduces a few migration issues.

7.2. NEW FEATURES

Client proxy support for java.io.Closeable interface

When you close a client proxy using the java.io.Closeable interface (which is supported, for example, by the **JaxWsClientProxy** class), *any open request/response sequences are now terminated* For example, if **proxy** is an instance of **JaxWsClientProxy**, you can close the proxy as follows:

```
((java.io.Closeable)proxy).close()
```

7.3. PACKAGING OF BUNDLES/JARS

Refactored cxf-api and cxf-rt-core bundles

The **cxf-api** bundle and the **cxf-rt-core** bundle are no longer available in Apache CXF 3.0.x. These bundles have been refactored as follows: the **cxf-api** bundle content has been split up, with the Java API moving to the new **cxf-core** bundle and the WSDL interfaces moving to the new **cxf-rt-wsdl** bundle; the **cxf-rt-core** bundle content is now provided in the new **cxf-core** bundle (which also now incorporates the Java API).

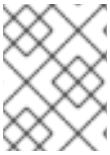
These changes are summarized in Table 7.1, “Equivalent bundles for cxf-api and cxf-rt-core” .

Table 7.1. Equivalent bundles for cxf-api and cxf-rt-core

2.7.0 Bundles	3.0.x Bundles (equivalent)
cxf-api	**cxf-rt-wsdl** **cxf-core**
cxf-rt-core	**cxf-core**

DynamicClientFactory class

The **DynamicClientFactory** has been moved from the JAXB data binding bundle, **cxf-rt-databinding-jaxb**, to the Simple frontend bundle, **cxf-rt-frontend-simple**.



NOTE

You are strongly encouraged to use the **`JaxWsDynamicClientFactory`** subclass instead of **`DynamicClientFactory`**.

New cxf-rt-rs-client bundle

JAX-RS 2.0 Client API and CXF specific **`WebClient`** and proxy client code is now made available in the new **`cxf-rt-rs-client`** bundle. See also the section called “http://cxf.apache.org/jaxrs namespace”.

7.4. XML SCHEMA NAMESPACES

http://cxf.apache.org/jaxrs namespace

For JAX-RS client endpoints, the **`http://cxf.apache.org/jaxrs`** namespace has been changed to **`http://cxf.apache.org/jaxrs-client`** in Apache CXF 3.0.x. This namespace change affects the **`jaxrs:client`** element, but the **`jaxrs:server`** and **`jaxrs:model`** continue to use the **`http://cxf.apache.org/jaxrs`** namespace, as before.

7.5. API CHANGES

Removed classes

The following classes and interfaces have been removed in Apache CXF 3.0.x:

```
org.apache.cxf.bus.CXFBusImpl
org.apache.cxf.databinding.BaseDataReader
org.apache.cxf.databinding.BaseDataWriter
org.apache.cxf.transports.http.QueryHandler
org.apache.cxf.transports.http.StemMatchingQueryHandler
org.apache.cxf.transports.http.internal.QueryHandlerRegistryImpl
org.apache.cxf.helpers.XMLUtils
org.apache.cxf.ws.addressing.impl.AddressingPropertiesImpl
org.apache.cxf.common.xmlschema.XmlSchemaConstants
org.apache.cxf.common.util.SOAPConstants
org.apache.cxf.frontend.MethodDispatcher
org.apache.cxf.jaxb.JAXBToStringBuilder
org.apache.cxf.jaxb.JAXBToStringStyle
org.apache.cxf.interceptor.URIMappingInterceptor
org.apache.cxf.jaxrs.ext.form.Form
org.apache.cxf.jaxrs.ext.ParameterHandler
org.apache.cxf.jaxrs.ext.RequestHandler
org.apache.cxf.jaxrs.ext.ResponseHandler
```

org.apache.cxf.Bus.run() method

The **`org.apache.cxf.Bus.run()`** method has been removed, because it is no longer used.

org.apache.cxf.transport.Destination.getBackChannel method

The two unused parameters of the **org.apache.cxf.transport.Destination.getBackChannel** method (normally passed as null) have been removed. That is, in previous versions, the **getBackChannel** method had the following signature:

```
Conduit
getBackChannel(Message inMessage, Message unused1, EndpointReferenceType
unused2)
```

In version 3.0.x, the **getBackChannel** method now has the following signature:

```
Conduit
getBackChannel(Message inMessage)
```

Spring XML files

All files of the form **/META-INF/cxf/cxf-extension-*ExtensionName*.xml** have been removed in version 3.0.x. At one time, it was necessary to **xi:include** such files into your Spring XML file, in order to use a particular extension feature. These files have not been needed and have been deprecated for a long time.

org.apache.cxf.transport.ConduitInitiator and DestinationFactory interfaces

Some of the methods of the **org.apache.cxf.transport.ConduitInitiator** and **DestinationFactory** interfaces now take an additional parameter of type, **org.apache.cxf.Bus**. In particular, **ConduitInitiator** defines the following method signatures in version 3.0.x:

```
Conduit getConduit(EndpointInfo targetInfo, Bus bus)
throws IOException;

Conduit getConduit(EndpointInfo localInfo, EndpointReferenceType target,
Bus bus)
throws IOException;

Set<String> getUriPrefixes();
List<String> getTransportIds();
```

And **DestinationFactory** defines the following method signatures in version 3.0.x:

```
Destination getDestination(EndpointInfo ei, Bus bus)
throws IOException;

Set<String> getUriPrefixes();

List<String> getTransportIds();
```

bus-extensions.xml file

When implementing an extension to the CXF Bus, the **bus-extensions.xml** file is no longer supported. Instead, use the approach of defining a **bus-extensions.txt** file. In the JAR file that implements your Bus extension, create the file, **META-INF/cxf/bus-extensions.txt**, and in this file list the classname of the class that implements the extension. The CXF Bus will automatically instantiate your extension class when it is needed.

Refactored XML parsing utilities

All code for StAX-based XML parsing and writing has been moved into the **`org.apache.cxf.staxutils.StaxUtils`** class and DOM-based utility code to the **`org.apache.cxf.helpers.DOMUtils`** class. The **`org.apache.cxf.helpers.XMLUtils`** class has been eliminated.

AddressingProperties interface is now a class

The **`org.apache.cxf.ws.addressing.AddressingProperties`** interface has been turned into a concrete class, which can be created directly using **`new`**. The **`AddressingPropertiesImpl`** class has been removed.

Policy classes in the org.apache.cxf.ws.policy package

Many of the classes in the **`org.apache.cxf.ws.policy`** package, such as **`AlternativeSelector`** and **`PolicyEngine`**, have modified method signatures, which now pass the current **`Message`** as an additional parameter (where appropriate). This makes it possible to use the message context to select a policy alternative. However, you must keep in mind that the selected alternative is probably cached and therefore, if the contextual information changes, the alternative might not be recalculated.

New supportNotAvailableErrorsOnly property on FailoverTargetSelector

By default, the **`org.apache.cxf.clustering.FailoverTargetSelector`** class triggers failover only for HTTP 404 and HTTP 503 status errors. If you would prefer to fail over for all HTTP errors, set the new **`supportNotAvailableErrorsOnly`** property to **`false`**.

ServletController not overriding endpoint addresses by default

The **`org.apache.cxf.transport.servlet.ServletController`** class does not override endpoint addresses by default, because this would have unwanted side-effects, if an endpoint is accessed through multiple paths. If you would like to revert to the old behaviour, set **`disable-address-updates`** parameter to **`false`** on the **`CXFServlet`** class.

SchemaValidation annotation

The **`enabled`** property (previously deprecated) has been removed from the **`org.apache.cxf.annotations.@SchemaValidation`** annotation. Use the **`@SchemaValidation.type`** property to control the validation instead.

FactoryType annotation

In the **`org.apache.cxf.annotations.@FactoryType`** annotation, the **`FactoryType.Type`** enumeration no longer includes the value, **`Spring`**. In version 3.0.x, to select Spring, set **`factoryClass=SpringBeanFactory.class`**.

org.apache.cxf.jaxrs.ext.form.Form class

The **`org.apache.cxf.jaxrs.ext.form.Form`** has been removed in version 3.0.x. Use the JAX-RS 2.0 **`org.apache.cxf.jaxrs.ext.form.Form`** class instead. For example, you would replace the following (obsolete) JAX-RS 1.0 code:

```
// JAX-RS 1.0 (obsolete)
import org.apache.cxf.jaxrs.ext.form.Form;
...
Form form = new Form().set("a", "b");
```

With the following JAX-RS 2.0 compliant code:

```
// JAX-RS 2.0
import javax.ws.rs.core.Form;
...
Form form = new Form().param("a", "b");
```

org.apache.cxf.jaxrs.ext.ParameterHandler<T> interface

The **org.apache.cxf.jaxrs.ext.ParameterHandler<T>** interface has been removed in Apache CXF 3.0.x. Use the **javax.ws.rs.ext.ParamConverterProvider** class instead, which can be used both on the client side and on the server side.

org.apache.cxf.jaxrs.ext.RequestHandler and ResponseHandler interfaces

The **org.apache.cxf.jaxrs.ext.RequestHandler** and **ResponseHandler** filters have been removed in Apache CXF 3.0.x. Use the **javax.ws.rs.container.ContainerRequestFilter** and **ContainerResponseFilter** interfaces instead.

For example, to define your own **CustomerRequestFilter** and **CustomResponseFilter** filter classes, you could write code like the following:

```
// Java
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.container.ContainerResponseFilter;
...
public class CustomRequestFilter implements ContainerRequestFilter {
    public void filter(ContainerRequestContext context) {
        ...
    }
}

public class CustomResponseFilter implements ContainerResponseFilter {
    public void filter(ContainerRequestContext inContext,
ContainerResponseContext outContext) {
        ...
    }
}
```

JaxWsClientProxy.getClient(proxy) and ClientProxy.getClient(proxy)

It is no longer necessary to call the **getClient(proxy)** method on the **org.apache.cxf.jaxws.JaxWsClientProxy** class (for the JAX-WS front-end) or the **org.apache.cxf.frontend.ClientProxy** class (for the simple front-end) in order to convert a proxy object to a **org.apache.cxf.endpoint.Client** object. Because the proxies now implement the **Client** interface directly, you can simply cast a proxy object directly to a **Client** object.

javax.annotation.Resource annotation

The **javax.annotation.Resource** annotation can no longer be used to annotate JAX-RS context properties. For the JAX-RS binding, only the **javax.ws.rs.core.Context** annotation is supported from now on.

7.6. DEPENDENCIES

javax.mail dependency

The direct dependency on a **javax.mail** implementation has been removed and the Apache CXF Maven POM files do not pull one in transitively in version 3.0.x. For most users, this will not be a problem. However, if your application uses MTOM or Soap with Attachments (or similar feature) that requires some of the **DataContentHandler** classes from the mail implementations, you might need to add the dependency to your classpath or Maven POM file.

cxf-bundle no longer supported

In the Apache CXF versions prior to 2.6, it was possible to include *all* of the Apache CXF modules in your application by adding a dependency on the **cxf-bundle** artifact to your Maven **pom.xml** file—for example:

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-bundle</artifactId>
            <version>...</version>
        </dependency>
        ...
    </dependencies>
</project>
```

From Apache CXF 2.6 onwards, however, the **cxf-bundle** artifact *is no longer available*. Instead of adding a single dependency on the **cxf-bundle** artifact, it is now necessary to add dependencies for each of the individual Apache CXF modules that your application depends on. For example, the basic set of Maven dependencies you would need for a simple JAX-WS Java application is as follows:

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>3.0.4.redhat-620133</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>3.0.4.redhat-620133</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
```

```
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>3.0.4.redhat-620133</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>3.0.6.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

7.7. MISCELLANEOUS CHANGES

Changed behaviour when providing WSDL location

If the WSDL location passed into Apache CXF is not valid (through the Java API or the XML API), previous versions of Apache CXF could in some cases ignore the error and proceed as if `null` was passed for the WSDL location. Apache CXF 3.0 now always throws an exception, if the WSDL location is invalid.