Red Hat JBoss Fuse 6.1

Migration Guide

Migrating to Red Hat JBoss Fuse 6.1

Last Updated: 2017-10-12

Red Hat JBoss Fuse 6.1 Migration Guide

Migrating to Red Hat JBoss Fuse 6.1

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

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 6.0 to JBoss Fuse 6.1. 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 6.1 have been upgraded. The following versions are used in JBoss Fuse:

Table 1.1. Component Versions

Component	Version for 6.0	Version for 6.1
Apache ActiveMQ	5.8.0	5.9.x
Apache Camel	2.10.0	2.12.x
Apache CXF	2.6.0	2.7.x
Apache Karaf	2.3.0	2.3.x
Fabric8 (was Fuse Fabric)	7.2.0	1.0.0
Spring framework	3.1.3	3.2.x

Fuse Management Console

In JBoss Fuse 6.1, the implementation of the Fuse Management Console has changed and it is now based on the open source Hawt.io project. The new Fuse Management Console has a radically different look-and-feel from the old version, with greatly expanded functionality and many new features.

When you start up the JBoss Fuse container (for example, by running the command, **`bin/fuse`**), the new Fuse Management Console is installed by default. You can access it by navigating to the following URL in your browser:

```
http://localhost:8181
```

For more details, see "Management Console User Guide".

NOTE

The ActiveMQ Web console has also been removed from the JBoss Fuse 6.1 product. Equivalent functionality for Apache ActiveMQ is now provided by the Fuse Management Console.

Fuse IDE

The Fuse Integrated Development Environment (IDE) is no longer provided as a standalone Eclipse executable. Fuse IDE has been renamed to *Fuse Plugins* for JBoss Developer Studio. To install Fuse Plugins, first download and install JBoss Developer Studio and then install the relevant Eclipse plug-ins. For detailed instructions, see "Installation Guide".

ActiveMQ resource adapter on JBoss EWS, JBoss EAP, and JBoss Fuse Service Works

The ActiveMQ resource adapter has been certified to function correctly on the JBoss Enterprise Web Services container, the JBoss Enterprise Application Platform container, and on JBoss Fuse Service Works, enabling you integrate JMS applications running on those containers with JBoss A-MQ.

Apache Camel deployed on JBoss EWS and JBoss EAP

The Apache Camel core library has been certified to function correctly on the JBoss Enterprise Web Services container and on the JBoss Enterprise Application Platform container.

Documentation reorganization

Some titles in the JBoss Fuse documentation library have changed and some books have been consolidated, so that there are slightly fewer books than before. In particular, the following books are new:

Management Console User Guide

Covers the new Hawt.io based management console, replacing the old *Using the Management Console* book.

Fabric Guide

A reference for Fuse Fabric technology.

Apache Camel Development Guide

Consolidates the various Camel development guides (with the exception of the *Apache Camel Component Reference*) into a single book.

Apache CXF Development Guide

Consolidates the various CXF development guides into a single book.

JBI Development Guide

Consolidates all of the Java Business Integration (JBI) guides into a single book.

Apache ActiveMQ changes

The changes resulting from the upgrade to version 5.9.0 are detailed in Chapter 2, *Apache ActiveMQ Issues*.

The most important changes are:

- The ActiveMQ Web console is no longer supported.
- If you are using a JDBC store with a broker that uses XA transactions, you will need to make some changes to the database schema, when migrating the database store.
- Dynamic configuration updates are now supported on a standalone broker.
- A generic JMS XA connection pool library is now available, which makes it possible to deploy third-party JMS providers.

Apache Camel changes

The changes resulting from the upgrade to version 2.12.0 are detailed in Chapter 3, *Apache Camel Issues*.

The most important changes are:

- Apache MINA components are deprecated (both MINA and MINA2).
- In Spring Web Services, the deprecated built-in **`CommonsHttpMessageSender`** has been replaced by **`HttpUrlConnectionMessageSender`**. Your Camel applications might be affected, if they use either the **`timeout`** option or the **`sslContextParameters`** option of this component in a producer endpoint.
- The **`camel-scala-29`** module has been removed.
- Spring version 3.0.x and 3.1.x are not supported in this release.

Apache Karaf Web console

The Karaf Web console is no longer supported in JBoss Fuse 6.1. You can use the new Hawtio-based Fuse Management Console instead, which has the same functionality (and more) as the Karaf Web console.

1.2. MIGRATING SPRING-DM TO BLUEPRINT

Spring-DM is now deprecated

Spring Dynamic Modules (Spring-DM) is now *deprecated* and will be removed from a future release of JBoss Fuse. You can continue to use Spring XML and the Spring framework, however, as long as you do not invoke the Spring-DM component.

Prefer Blueprint over Spring-DM

The Blueprint container is now the preferred framework for instantiating, registering, and referencing OSGi services, because this container has now been adopted as an OSGi standard. This ensures greater portability for your OSGi service definitions in the future.

Spring Dynamic Modules (Spring-DM) provided much of the original impetus for the definition of the Blueprint standard, but should now be regarded as obsolescent. Using the Blueprint container does *not* prevent you from using the Spring framework: the latest version of Spring is compatible with Blueprint.

How to tell whether your code uses Spring-DM

In Spring XML files, the Spring-DM component is associated with the following XML namespace:

```
http://www.springframework.org/schema/osgi
```

To identify the parts of your application that use Spring-DM, search for the preceding namespace string in your code.

How to migrate Spring-DM to Blueprint

If you have a Spring XML file that uses the Spring-DM component, migrate this file to Blueprint XML, as follows:

1. In the standard Maven project layout, Blueprint XML files are stored under the following directory:

   ```
   src/main/resources/OSGI-INF/blueprint
   ```

   If it does not already exist, create this directory under your Maven project.

2. Move the relevant Spring XML file from the Spring directory, **`src/main/resources/META-INF/spring`**, to the Blueprint directory, **`src/main/resources/OSGI-INF/blueprint`**.

3. Edit the Spring XML file in order to convert it to a Blueprint XML file. For example, a typical Spring XML file using Spring-DM has the following outline:

   ```
   <beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:osgi="http://www.springframework.org/schema/osgi">
       ...
       <osgi:reference id="osgiPlatformTransactionManager"

   interface="org.springframework.transaction.PlatformTransactionManage
   r"/>

       <osgi:reference id="osgiJtaTransactionManager"

   interface="javax.transaction.TransactionManager"/>
       ...
   </beans>
   ```

   You can convert this Spring XML file to a Blueprint XML file by replacing the **`beans`** root element by a root **`blueprint`** root element, and replacing Spring-DM **`osgi:reference`** elements by Blueprint **`reference`** elements. For example:

   ```
   <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       ...
       <reference id="osgiPlatformTransactionManager"
   ```

```
interface="org.springframework.transaction.PlatformTransactionManage
r"/>

    <reference id="osgiJtaTransactionManager"
               interface="javax.transaction.TransactionManager"/>
    ...
</blueprint>
```

1.3. FABRIC MIGRATION

Overview

If you use Fuse Fabric containers to deploy your applications, you should read this section to understand the issues that affect migration to Red Hat JBoss Fuse. If you use standalone containers, however, you can ignore this section.

Fabric package has changed

In JBoss Fuse 6.1 and JBoss A-MQ 6.1, the Fabric package name has changed from **`org.fusesource.fabric`** to **`io.fabric8`**, which has the following migration impact:

- Configuration PIDs for Fabric have been renamed from **`org.fusesource.fabric.`*`name`*** to **`io.fabric8.`*`name`***. For example, this affects the names of the following configuration files under the **`etc/`** directory:

  ```
  io.fabric8.datastore.cfg
  io.fabric8.fab.osgi.url.cfg
  io.fabric8.maven.cfg
  ```

- If you use any Fabric APIs directly in your own application code, you must replace references to the **`org.fusesource.fabric`** package name by **`io.fabric8`**.

- If your Maven **`pom.xml`** files have any direct dependencies on Fabric artifacts, you need to change the Maven GroupId from **`org.fusesource.fabric`** to **`io.fabric8`** (as well as updating the dependency version). For example, the following **`fabric-zookeeper`** dependency:

  ```
  <dependency>
      <groupId>org.fusesource.fabric</groupId>
      <artifactId>fabric-zookeeper</artifactId>
      <version>7.2.0</version>
  </dependency>
  ```

  Should be replaced by the following dependency:

  ```
  <dependency>
      <groupId>io.fabric8</groupId>
      <artifactId>fabric-zookeeper</artifactId>
      <version>1.0.0</version>
  </dependency>
  ```

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>6.1.0.redhat-379</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>6.1.0.redhat-
379</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>2.12.0.redhat-610379</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-blueprint</artifactId>
 <version>2.12.0.redhat-610379</version>
    </dependency>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-jetty</artifactId>
 <version>2.12.0.redhat-610379</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. APACHE ACTIVEMQ ISSUES

Abstract

Red Hat JBoss Fuse 6.1.0.redhat-379 uses Apache ActiveMQ 5.9.0. Since the last release, Apache ActiveMQ has been upgraded from version 5.8.0 to version 5.9.0. This introduces a few migration issues.

2.1. KEY MIGRATION ISSUES

ActiveMQ Web console no longer supported

The Apache ActiveMQ Web console is no longer supported in JBoss A-MQ 6.1 and JBoss Fuse 6.1. It is recommended that you use the Fuse Management Console instead (installed by default in the container). You can access the new Hawtio-based Fuse Management Console by navigating to the **`http://localhost:8181`** URL in your browser.

Migrating a JDBC store with XA transactions

If you are using the JDBC persistence adapter with XA transactions enabled, you should note that the database schema for the JDBC store has changed, going from version 5.8.0 to version 5.9.0. If you need to migrate the contents of an existing JDBC store to ActiveMQ 5.9.0, you must perform the following steps:

1. Note that this change to the database schema only affects JDBC stores that used with XA transactions. If your existing broker does not use XA transactions, there is no need to perform these migration steps.
2. If the old broker (version 5.8.0) is currently running, make sure no transactions are in flight before attempting to stop the broker.
3. Stop the broker.
4. Alter the database tables by executing database commands similar to the following (you will probably need to adapt these commands to suit the syntax of your particular database):

```
ALTER TABLE ACTIVEMQ_MSGS ALTER COLUMN XID VARCHAR(250)
CREATE INDEX ACTIVEMQ_MSGS_XIDX ON ACTIVEMQ_MSGS (XID)
ALTER TABLE ACTIVEMQ_ACKS ALTER COLUMN XID VARCHAR(250)
CREATE INDEX ACTIVEMQ_ACKS_XIDX ON ACTIVEMQ_ACKS (XID)
```

5. Configure the new broker (version 5.9.0) to use the JDBC store.
6. Start the new broker.

2.2. 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 Red Hat 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.

2.3. NEW FEATURES

Dynamic configuration updates

JBoss A-MQ now has a runtime configuration plug-in, which enables you to update the configuration of a standalone broker (that is, a broker that is not part of a fabric) dynamically at run time.

For details, see the section "Modifying a Running Standalone Broker's XML Configuration" in the chapter "Editing a Broker's Configuration" in the *Managing and Monitoring a Broker* guide from the JBoss A-MQ library.

Generic JMS XA connection pool

The **`activemq-jms-pool`** component is a new library that provides generic JMS connection pool classes that support the XA transaction protocol. Using this library, it is possible for third-party JMS providers to participate in XA transactions managed by any JTA transaction manager.

For details of the generic XA-aware connection pool library, see the chapter "XA Transactions in Red Hat JBoss Fuse" in the *Transaction Guide* from the JBoss Fuse library.

AMQP protocol

Red Hat JBoss Fuse 6.1 now supports the AMQP 1.0 protocol (OASIS Advanced Message Queuing Protocol). You can access the AMQP protocol using a URI of the form, **`amqp://`*`Hostname`*`:`*`Port`***. SSL security is also supported.

Red Hat JBoss Fuse 6.1 includes the Apache Qpid client API, which provides a JMS client API that is compatible with AMQP. You can use this client API to implement AMQP clients in Java.

C, C++ and .Net client libraries for Openwire

Red Hat JBoss Fuse 6.1 includes the C, C++ (CMS) client library and the .Net (NMS) client library for Openwire (the native Apache ActiveMQ wire protocol).

2.4. FEATURE PREVIEWS

Technical previews of features in development

The following features are included in the JBoss Fuse 6.1 release in order to let you preview technology that is currently under development. *These features are not yet supported and are not suitable for deployment in a production environment.*

LevelDB store (on selected platforms)

Red Hat JBoss Fuse includes a technical preview of the new LevelDB store, which uses Google's LevelDB library to maintain the indexes in the log files. The LevelDB store supports XA transactions in ActiveMQ 5.9. The LevelDB store is available only on Windows, Linux, and Mac OS platforms.

NOTE

The LevelDB store is provided with native libraries on Linux only. On all of the other supported operating systems, LevelDB is provided as a Java implementation.

IMPORTANT

This is a technical preview only. The LevelDB store is currently not supported and is not suitable for deployment in a production environment.

2.5. SECURITY FIXES

SSL transport cipher suites

Fixed AMQ-4582, affecting the SSL transport cipher suites. Previously, if you specified an invalid cipher suite to the `transport.enabledCipherSuites` parameter on an SSL transport connector, the broker would start with all ciphers enabled.

JAAS authorization now compatible with Karaf JAAS authentication

The implementation of the JAAS authorization plug-in has been modified so that it is compatible with the Apache Karaf JAAS authentication module. This makes it possible to integrate the JAAS authorization plug-in with the Karaf JAAS authentication module when the broker is deployed in an OSGi container. For more details, see the *JBoss A-MQ Security Guide*.

Allow Bouncy Castle security provider to be used

Fixed AMQ-4520, which is caused by a bug in the default SSL provider that comes with Java 7 (affecting the Diffie-Hellman cipher suite). You can now work around this issue by adding the Bouncy Castle security producer to the Java 7 `lib` directory.

Removed command agent

Removed the command agent, which is no longer needed and might potentially have exposed a security hole through the JMS protocol.

2.6. PERSISTENCE AND FAILOVER

Recommended message stores

If you are using one of the deprecated message stores, it is recommended that you migrate to one of the following message stores instead:

- KahaDB message store.
- JDBC message store.

Multiple stores for different destinations

See AMQ-4310.

2.7. TRANSPORT PROTOCOL CHANGES

XMPP transport

The XMPP transport is now *deprecated* and will be removed in a future release.

Openwire over UDP

The Openwire over UDP protocol combination is now *deprecated* and will be removed in a future release.

2.8. DEPENDENCY UPGRADES

Spring framework

JBoss A-MQ and JBoss Fuse use Spring framework version 3.2.x. Spring version 3.0.x and version 3.1.x *not* supported in this release.

Apache Karaf

JBoss A-MQ and JBoss Fuse use Apache Karaf version 2.3.0.

Jasypt

In ActiveMQ 5.9.0, Jasypt is upgraded to version 1.9.1.

Jolokia

In ActiveMQ 5.9.0, Jolokia is upgraded to version 1.1.4.

Apache Xerces

In ActiveMQ 5.9.0, Apache Xerces is upgraded to version 2.11.0.

json-simple

In ActiveMQ 5.9.0, **`json-simple`** is upgraded to version 1.1.1.

Apache Derby

In ActiveMQ 5.9.0, Apache Derby is upgraded to version 10.10.1.1.

Apache commons-io

In ActiveMQ 5.9.0, Apache commons-io is upgraded to version 2.4.

Apache Qpid

In ActiveMQ 5.9.0, Apache Qpid is upgraded to version 0.26.

2.9. API CHANGES

isSlave method reinstated

In Apache ActiveMQ 5.9.0, reinstated the **`isSlave`** method on the **`org.apache.activemq.broker.BrokerService`** class.

PooledConnectionFactory

Renamed the **`setTimeBetweenExpirationCheckMillis`** method to **`getTimeBetweenExpirationCheckMillis`** in the **`org.apache.activemq.jms.pool.PooledConnectionFactory`** class.

New activemq-spring module

In ActiveMQ 5.9.0, the code base has been refactored to localize dependencies on the Spring framework, which are now packaged in the new **`activemq-spring`** module.

CHAPTER 3. APACHE CAMEL ISSUES

3.1. SUMMARY OF APACHE CAMEL 2.10 TO APACHE CAMEL 2.12 MIGRATION

Overview

Red Hat JBoss Fuse 6.1.0.redhat-379 uses Apache Camel 2.12. Since the last release, Apache Camel has been upgraded from version 2.10 to version 2.12. 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.

You should also note that Maven 3.0.4 or better is required to build the source.

Apache MINA components are deprecated

In Apache Camel 2.12, both the MINA component and the MINA2 component are deprecated. Use the Camel Netty component instead.

New Camel components

Many new Camel components have been added in Apache Camel versions 2.11 and 2.12. In particular, note the following new components:

SAP component

The SAP Component enables outbound and inbound communication to and from SAP systems using synchronous remote function calls, sRFC. The component uses the SAP Java Connector (SAP JCo) library to facilitate bidirectional communication with SAP. The component supports two types of endpoints: destination endpoints and server endpoints.

SAP Netweaver component

The SAP Netweaver component integrates with the SAP NetWeaver Gateway using HTTP transports.

Netty HTTP component

The Netty HTTP component is an extension of the Netty component that can be used to create HTTP sockets with Netty.

Yammer component

The Yammer component enables you to interact with the Yammer enterprise social network.

Facebook component

The Facebook component provides access to all of the Facebook APIs accessible using Facebook4J.

3.2. SPRING FRAMEWORK

Spring version

Since Apache Camel 2.12, the version of Spring that you can use with Apache Camel *must* be 3.1 (or later). Spring 3.0.x is 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 3.1 Spring schema.

The Spring 3.1 schema is located at the following Web page:

```
http://www.springframework.org/schema/beans/spring-beans-3.1.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-
3.1.xsd">
```

Spring 3.1 new features

For a summary of the new features in Spring 3.1, see New Features and Enhancements in Spring 3.1.

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.

3.3. PRODUCT DEPENDENCIES

Java development kit

Since Apache Camel 2.7, the Apache Camel requires at least JDK 1.6 and also supports JDK 1.7.

Maven version

Since Apache Camel 2.11, you require Maven 3.0.4 or later to build the source distribution of Apache Camel.

Apache CXF version

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

Jetty version

Since Apache Camel 2.10, the Jetty component is updated to Jetty 7.5.4.

Component dependencies

In Apache Camel 2.11, some components have had their third party dependencies upgraded, as follows:

- Aries Blueprint 0.3 to 1.0.1
- Async Http Client 1.7.5 to 1.7.13
- Avro 1.6.2 to 1.7.3
- AWS 1.3.10 to 1.3.27
- BeanIO 2.0.0 to 2.0.5
- Bouncycastle 1.46 to 1.47
- Commons CSV 1.0-r706900_3 to 1.0-r706899_5
- Commons Exec 1.0.1 to 1.1
- Commons Logging 1.1.1 to 1.1.2
- Commons Net 3.1 to 3.2
- ConcurrentLinkedHashMap 1.2 to 1.3.2
- Castor 1.3.1 to 1.3.2
- CXF 2.6.5 to 2.7.4
- Dozer 5.3.2 to 5.4.0
- Ehcache 2.5.1 to 2.5.2
- Google App Engine 1.6.6 to 1.7.4
- Google Guava 13.0.1 to 14.0.1
- Groovy 1.8.6 to 2.1.3
- GSon 2.1 to 2.2.2
- Guice 2.0 to 3.0
- Hadoop 1.0.4 to 1.1.1
- Hazelcast 2.0.2 to 2.5
- Hibernate 4.1.9 to 4.1.11
- HBase 0.90.5 to 0.94.3
- HTTP Client 4.1.3 to 4.2.3
- HTTP Core 4.1.4 to 4.2.3
- Icu4j 4.0.1 to 4.8.1.1

- Jackson 1.9.7 to 2.1.14
- Jaxen 1.1.3 to 1.1.4
- JClouds 1.4.0 to 1.5.7
- Jettison 1.3.1 to 1.3.3
- Jetty 7.5.4 to 7.6.8
- JRuby 1.6.7 to 1.7.2
- JT400 6.0 to 6.7
- Jython 2.2.1 to 2.5.3
- Krati 0.4.5 to 0.4.8
- LevelDBJNI 1.2 to 1.6
- Lucene 3.6.0 to 3.6.1
- Mina 2.0.4 to 2.0.7
- MongoDB Java Driver 2.7.3 to 2.9.1
- MQTTClient 1.2 to 1.4
- MyBatis 3.1.1 to 3.2.2
- Netty 3.5.1 to 3.6.5
- Ognl bundle 3.0.4_1 to 3.0.5_1
- OSGi 4.2.0 to 4.3.1
- Pax Logging 1.5.3 to 1.6.10
- QPid 0.16 to 0.20
- Quartz 1.8.5 to 1.8.6
- Quickfix 1.5.2 to 1.5.3
- Restlet 2.0.14 to 2.0.15
- Saxon 9.3.0.11 to 9.4.0.1
- Scala 2.9.1 to 2.10.1
- ServiceMix Specs 1.9.0 to 2.2.0
- Shiro 1.2.0 to 1.2.1
- SLF4J 1.6.6 to 1.7.5
- Solr 3.6.0 to 3.6.1

- Spring Batch 2.1.8 to 2.1.9
- Spring Framework 3.1.1 to 3.1.4
- Spring Integration 2.1.2 to 2.2.3
- Spring Security 3.1.0 to 3.1.3
- SSHD 0.6.0 to 0.8.0
- StringTemplate 3.2.1 to 4.0.2
- TestNG 6.0.1 to 6.8
- Twitter4j 2.2.5 to 3.0.3
- Woodstox 4.1.2 to 4.2.0
- Xbean-Spring 3.12 to 3.13
- Xerces 2.9.1 to 2.10.0
- XmlBeans 2.5.0 to 2.6.0
- XStream 1.4.2 to 1.4.3
- Zookeeper 3.3.5 to 3.4.5

In Apache Camel 2.12, some components have had their third party dependencies upgraded, as follows:

- AHC 1.7.11 to 1.7.19
- AWS Java SDK 1.3.27 to 1.5.1
- Beanio from 2.0.5 to 2.0.6
- Bouncy Castle 1.47 to 1.49
- Commons Codec 1.6 to 1.8
- Commons Net 3.1 to 3.3
- Commons Httpclient 4.2.3 to 4.2.5
- Commons Httpcore 4.2.3 to 4.2.4
- CXF 2.7.4 to 2.7.6
- Disruptor 3.1.1 to 3.2.0
- Ehcache 2.5.2 to 2.7.2
- GAE 1.7.4 to 1.8.2
- HAPI 2.0 to 2.1
- Groovy 2.1.3 to 2.1.6

- Hadoop 1.1.1 to 1.2.0
- HBase 0.94.6 to 0.94.10
- Hawtdispatch 1.13 to 1.17
- Hibernate Validator 4.1.0.Final to 5.0.1.Final
- Hibernate 4.2.3 to 4.2.4
- Jackson 2.1.4 to 2.2.2
- Jersey 1.13 to 1.17.1
- Jettison 1.3.3 to 1.3.4
- JRuby 1.7.2 to 1.7.4
- Krati 0.4.8 to 0.4.9
- Leveldbjni 1.6 to 1.7
- Mail 1.4.5 to 1.4.7
- MongoDB Java Driver 2.9.1 to 2.11.2
- MQTT Client 1.4 to 1.5
- MVEL 2.1.3.Final to 2.1.6.Final
- Netty 3.6.5 to 3.6.6
- OpenEJB 4.5.1 to 4.5.2
- OpenJPA 2.2.1 to 2.2.2
- Saxon 9.4.0.4 to 9.5.0.2
- Scala 2.10.1 to 2.10.2
- Shiro 1.2.1 to 1.2.2
- SNMP4J 2.1.0 to 2.2.2
- Spring 3.1.4.RELEASE to 3.2.4.RELEASE
- Spring Batch 2.1.9.RELEASE to 2.2.1.RELEASE
- Spring Data Redis 1.0.3.RELEASE to 1.0.4.RELEASE
- Spring Security 3.1.3.RELEASE to 3.1.4.RELEASE
- Spring Web Services 2.1.2 to 2.1.3
- TestNG 6.8 to 6.8.5
- XML Security (Santuario) 1.5.2 to 1.5.5.

- XStream 1.4.3 to 1.4.4

3.4. API CHANGES

API changes

The following changes have been made to the Java API:

HBase component

Since Apache Camel 2.11, renamed **`org.apache.camel.component.hbase.HBaseContats`** [sic] to **`org.apache.camel.component.hbase.HBaseConstants`**.

ManagementStrategy

Since Apache Camel 2.11, added getter/setter methods for the **`loadStatisticsEnabled`** property on **`org.apache.camel.spi.ManagementStrategy`**.

ExecutorServiceManager

Since Apache Camel 2.11, added **`shutdownGraceful`**, and **`awaitTermination`** methods to the **`ExecutorServiceManager`** class to make it easier to shut down a thread pool gracefully, while waiting for its tasks to complete.

Since Apache Camel 2.11, added the method **`newThread`** to **`org.apache.camel.spi.ExecutorServiceManager`** to create a new thread without using a thread pool.

org.apache.camel. Component interface

Since Apache Camel 2.11, added a new method, **`useRawUri`**, to the **`org.apache.camel.Component`** interface to allow components to use raw URIs when creating endpoints. (by default the URI has been encoded prior to creation).

GenericFileConsumer

Since Apache Camel 2.11, the **`isMatched`** method on **`GenericFileConsumer`** is abstract and has an extra parameter.

ManagedTracerBacklogMBean renamed

Since Apache Camel 2.12, the **`org.apache.camel.api.management.mbean.ManagedTracerBacklogMBean`** interface has been renamed to **`org.apache.camel.api.management.mbean.ManagedBacklogTracerMBean`**.

StreamCache

Since Apache Camel 2.12, added new **`inMemory`** and **`length`** methods to **`org.apache.camel.StreamCache`**.

ManagementNamingStrategy

Since Apache Camel 2.12, changed the parameter type, **`ProcessorDefinition`**, to **`NamedNode`**, on the **`getObjectNameForProcessor`** method in the **`org.apache.camel.spi.ManagementNamingStrategy`** interface.

TypeConverterRegistry

Since Apache Camel 2.12, added **`removeTypeConverter`** method to **`org.apache.camel.spi.TypeConverterRegistry`**.

TypeConverter

Since Apache Camel 2.12, added the **`allowNull`** method to the **`org.apache.camel.TypeConverter`** interface.

ShutdownStrategy

Since Apache Camel 2.12, added **`hasTimeoutOccurred`** method to the **`ShutdownStrategy`** interface.

ManagementNamingStrategy

Since Apache Camel 2.12, changed the parameter type, **`ProcessorDefinition`**, to **`NamedNode`** on the **`getObjectNameForProcessor`** method in the **`org.apache.camel.spi.ManagementNamingStrategy`** interface.

Removed classes

Since Apache Camel version 2.11, the following classes have been removed:

```
org.apache.camel.model.NodeFactory
org.apache.camel.component.gae.context.GaeSpringCamelContext
```

Since Apache Camel version 2.12, the following classes have been removed:

```
org.apache.camel.processor.UnitOfWorkProcessor
org.apache.camel.processor.ChildUnitOfWorkProcessor
org.apache.camel.processor.RouteContextProcessor
org.apache.camel.processor.RouteInflightRepositoryProcessor
org.apache.camel.processor.RoutePolicyProcessor
org.apache.camel.processor.interceptor.BacklogTracerInterceptor
```

Moved classes

Since Apache Camel version 2.11, the following classes have moved to a different package:

- **`HBaseConstants`** has moved from **`org.apache.camel.component.hbase`** to **`org.apache.camel`**.

Since Apache Camel version 2.12, the following classes have moved to a different package or renamed:

- **`ManagedTracerBacklogMBean`** has been renamed to **`ManagedBacklogTracerMBean`** in the **`org.apache.camel.api.management.mbean`** package.
-

3.5. COMPONENT UPDATES

HL7 component

Since Apache Camel version 2.11, upgraded to use Apache Mina 2.x.

Since Apache Camel version 2.11, the default value of the **`convertLFtoCR`** option of the HL7 MLLP codec has been changed from **`true`** to **`false`**. Likewise, the HL7 **`DataFormat`** and **`TypeConverter`** classes do not perform this conversion anymore.

File and FTP component

Since Apache Camel version 2.11, the type for *file last modified* header is changed from **`java.util.Date`** to **`long`**.

Since Apache Camel version 2.12, when using idempotent repository, the file and FTP components invoke the **`contains`** method for a file name, but not for a directory name.

CSV component

Since Apache Camel version 2.11, added support to parse a CSV, if a field begins with a **`'`** (single quote) character.

MINA2 component

Since Apache Camel version 2.11, MINA2 uses **`OrderedThreadPoolExecutor`** by default. There is a new option, **`orderedThreadPoolExecutor`**, which you can configure to **`false`**, in order to revert back to unordered. If using SSL, however, an ordered thread pool must be used.

Netty component

Since Apache Camel version 2.11, Netty uses **`OrderedThreadPoolExecutor`** by default. There is a new option, **`orderedThreadPoolExecutor`**, which you can configure to **`false`**, in order to revert back to unordered. If using SSL, however, an ordered thread pool must be used.

Since Apache Camel version 2.11, Netty requires **`commons-pool`** as a dependency.

Timer component

Since Apache Camel version 2.11, the default value of the **`delay`** option on the Timer component has changed from **`0`** to **`1000`**.

VM and SEDA components

Since Apache Camel version 2.11, for any VM (or SEDA) endpoints with the same name, if the endpoints specify a queue size, the queue size *must be the same*. The queue names and their corresponding queue size values are logged at INFO level, in order to help you spot any mis-configuration of the queue sizes.

ZooKeeperEndpoint

Since Apache Camel version 2.11, the getter and setter for the **`awaitExistence`** bean property from the **`org.apache.camel.component.zookeeper.ZooKeeperEndpoint`** class have been marked as **`@deprecated`** and removed from the API documentation (the property has no effect).

XSLT component

Since Apache Camel version 2.11, if you define an XSLT script that has **`xi:include`** elements, the included files are found by searching the same scope as the endpoint (by default). That is, if the endpoint URI is specified using the **`file:`** scheme, the included files are found by searching the file system; or if the endpoint is specified using the **`classpath:`** scheme, the included files are found by searching the classpath. You can override this behavior by specifying the **`file:`** or **`classpath:`** scheme explicitly in the **`href`** attribute of **`xi:include`**. Before Apache Camel version 2.11, the default behavior was to search the classpath for included files, irrespective of the scheme used in the endpoint URI.

Google App Engine (GAE) components

Since Apache Camel version 2.11, GAE components no longer have any dependency on the Spring framework.

XML Security component

Since Apache Camel version 2.12, if you want to support the RSA v1.5 key transport algorithm at run time, you must configure it explicitly. If not, requests to use this algorithm are rejected, by default.

FOP component

Since Apache Camel version 2.12, the **`userConfigURL`** option loads resources from the classpath by default.

3.6. MISCELLANEOUS CHANGES

Dead letter channel EIP

Since Apache Camel 2.11, setting **`allowRedeliveryWhileStopping=false`** on the dead letter channel EIP now moves the current message to the dead letter channel (when stopping), instead of rejecting the message.

Simple language

Since Apache Camel 2.11, the Simple language has limited its support for the escape character to only new line, carriage return and tab characters. This makes it easier to use back-slash characters in the Simple language without double escaping etc.

Thread pools

Since Apache Camel 2.11, if the **`maximumQueueSize`** property of Thread Pools or Thread Pool Profile has a value that is **`0`** or negative, this now means disable the worker queue, and use direct handover. If you want an unbounded queue size, set the **`maximumQueueSize`** to **`Integer.MAX_VALUE`**, which was what the code previously did.

Route ID validation

Since Apache Camel 2.11, explicitly assigned route IDs on Apache Camel routes are now validated as the route starts up, in order to ensure that route IDs are unique in the current **`CamelContext`**.

Graceful shutdown

Since Apache Camel 2.12, requires the timeout value to be positive.

Tracer

Since Apache Camel 2.12, the tracer must be explicitly enabled on **`CamelContext`** in order to be available in and in use.

Simple language

Since Apache Camel 2.12, the unary operators are only applied to functions.

Since Apache Camel 2.12, you can access an exception message in the XML DSL only as **`${exception.message}`**, not as **`${exception}`**. Hence, if you have any code like the following:

```
<setHeader headerName="DLQ_Reason">
 <simple>${exception}</simple>
</setHeader>
```

You must replace it by code like the following:

```
<setHeader headerName="DLQ_Reason">
 <simple>${exception.message}</simple>
</setHeader>
```

JMX

Since Apache Camel 2.12, you cannot use the JMX bean, **`org.apache.camel.api.management.mbean.ManagedSendProcessorMBean`**, to change the destination.

CHAPTER 4. APACHE CXF ISSUES

4.1. SUMMARY OF APACHE CXF 2.6.0 TO 2.7.0 MIGRATION

Overview

JBoss Fuse uses Apache CXF 2.7.0. Since the last release, Apache CXF has been upgraded from version 2.6.0 to version 2.7.0. This introduces a few migration issues.

Upgrading JAX-RS endpoints to JAX-RS 2.0 without recompiling

If you have compiled your application bundles against the JSR-311 (JAX-RS 1.1) API and would like to get them running in CXF 2.7.x without recompiling them against JSR-339 (JAX-RS 2.0) API, use one of the following approaches:

- Import the **`javax.ws.rs`**, **`javax.ws.rs.core`** and **`javax.ws.rs.ext`** bundles with the **`[1.1, 3)`** version range and deploy the **`org.apache.servicemix.specs.jsr339-api-m10-2.1.0`** specification bundle, or
- Deploy the **`org.apache.servicemix.specs.jsr339-api-m10-2.2.0`** specification bundle (when it is released).

4.2. NEW FEATURES

Netty HTTP transport

There is a new Netty HTTP transport, which can be used in place of the plain HTTP transport. To use the Netty transport, prefix a HTTP URL with **`netty://`**—for example, **`netty://http://localhost:8080/server`**. The Netty transport is provided in the following artifacts (for client side and server side respectively):

- **`cxf-rt-transports-http-netty-client`**
- **`cxf-rt-transports-http-netty-server`**

UDP transport

New UDP Transport.

HTTPAsyncClient-based HTTP transport

New optional HTTP transport based on Apache HTTP Components HttpAsyncClient.

SOAP over UDP

Support for the protocol, SOAP over UDP.

Message validation

Ability to apply schema validation selectively to incoming or outgoing messages, by setting the **`schema-validation-enabled`** property to **`IN`**, **`OUT`**, **`BOTH`**, or **`NONE`**. **`@SchemaValidationEnabled`** annotation updated with the **`type=IN|OUT|BOTH|NONE`** parameter.

WS-Discovery

Support for WS-Discovery, as follows:

- Services can send **`Hello`** or **`Bye`** when started or stopped and can also respond to probe requests.
- An API for sending probes and resolving to **`EndpointReferences`** is provided.

JAX-RS 2.0 specification

Initial support for parts of the JAX-RS 2.0 (JSR-339) specification, as follows:

- Additional methods on the **`WebClient`** class to provide asynchronous invocations.
- Support for new filters, interceptors, dynamic features, exception classes, and more.

4.3. API CHANGES

HTTPConduit class

The **`HTTPConduit`** class has been made **`abstract`** and the code relating to **`HttpURLConnection`** has been moved to **`URLConnectionHTTPConduit`**. Several method calls of the **`HTTPConduit`** class that used to take **`HttpURLConnection`** objects have been eliminated. Moreover, most methods taking a URL object now take a URI object instead. The **`HTTPConduit.WrappedOutputStream`** class is also now **`abstract`**.

If you have implemented custom subclasses of **`HTTPConduit`**, it is probably necessary to refactor them as a subclass **`URLConnectionHTTPConduit`**, instead of **`HTTPConduit`**.

VersionTransformer and MAPCodec classes

The **`VersionTransformer`** and **`MAPCodec`** classes (used for WS-Addressing) now use the header list on the **`SoapMessage`** directly, instead of encoding the WS-Addressing headers as DOM elements. The **`encode`** methods have therefore been changed to take a **`JAXBContext`** argument, instead of a **`Marshaller`** argument. Any custom **`VersionTransformers`** will need to be updated.

AbstractFeatures as return value or parameter

All methods with an argumnet or return value of type, **`org.apache.cxf.feature.AbstractFeatures`**, have been changed to use **`org.apache.cxf.feature.Feature`** instead.

Removed classes

The following classes have been removed from the JAX-RS library:

```
org.apache.cxf.jaxrs.client.ResponseReader
org.apache.cxf.jaxrs.client.ServerWebApplicationException
org.apache.cxf.jaxrs.client.ClientWebApplicationException
```

The first class in the list is not needed with the JAX-RS 2.0 **`Response`** class, while the last two classes are replaced by the classes, **`javax.ws.rs.WebApplicationException`** and **`javax.ws.rs.client.ClientException`**.

Security Token Service (STS) API

The STS Claims object now takes a **`List<String>`** argument for the claims values, instead of a single **`String`** value.

Deprecated APIs

The following CXF JAX-RS specific extensions have been deprecated and will eventually be removed:

org.apache.cxf.jaxrs.ext.ParamHandler

Use the **`javax.ws.rs.ext.ParamConverterProvider`** and **`javax.ws.rs.ext.ParamConverter`** classes instead.

org.apache.cxf.jaxrs.ext.RequestHandler

Use the **`javax.ws.rs.container.ContainerRequestFilter`** class instead.

org.apache.cxf.jaxrs.ext.ResponseHandler

Use **`javax.ws.rs.container.ContainerResponseFilter`** instead.

4.4. DEPENDENCIES

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>2.7.0.redhat-610379</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>2.7.0.redhat-610379</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http-jetty</artifactId>
            <version>2.7.0.redhat-610379</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>
```