Fuse ESB Enterprise

Migration Guide

Version 7.1

December 2012

Integration Everywhere

Migration Guide

Version 7.1

Updated: 08 Jan 2014

- HAPI-OSGI-Base Module (http://hl7api.sourceforge.net/hapi-osgi-base/) ca.uhn.hapi:hapi-osgi-base:bundle:1.2

  License: Mozilla Public License 1.1 (http://www.mozilla.org/MPL/MPL-1.1.txt)

Table of Contents

List of Tables

Chapter 1. Runtime Changes

In Fuse ESB Enterprise version 7.1, the Fuse ESB has been upgraded to Apache Karaf 2.3.0.

JDK version

Fuse ESB Enterprise 7.1 supports Java 6 and Java 7.

Important

Java 5 is no longer supported.

Maven version

Fuse ESB Enterprise 7.1 requires Maven 3.

Important

Maven 2.x is *not* supported.

Saxon version

Fuse ESB Enterprise 7.1's Saxon version was updated from 9.1.0.8 to 9.4 HE.

This update makes significant changes to the feature set of the Saxon library. The most significant of these changes is that "extensibility using reflexion"—calling static methods from a Java class—is no longer supported.

For a full list of changes see http://www.saxonica.com/feature-matrix.html.

BPEL support

Fuse ESB Enterprise 7.1 does **not** support Apache ODE. The latest version of ODE is not compatible with Fuse ESB Enterprise.

Apache ActiveMQ

Fuse ESB Enterprise 7.1 supports Apache ActiveMQ 5.7.0.fuse-71-047. For information on the migration issues involved with Apache ActiveMQ see "Apache ActiveMQ Issues" on page 13.

Apache Camel

Fuse ESB Enterprise 7.1 supports Apache Camel 2.10.0.fuse-71-047. For information on the migration issues involved with Apache ActiveMQ see "Apache Camel Issues" on page 23.

Apache CXF

Fuse ESB Enterprise 7.1 supports Apache CXF 2.6.0.fuse-71-047. For information on the migration issues involved with Apache ActiveMQ see "Apache CXF Issues" on page 69.

OSGi framework

Fuse ESB Enterprise only ships with Apache Felix OSGi. Equinox is no longer included.

Upgraded kernel and dependencies

In Fuse ESB Enterprise 7.1, the kernel has been upgraded to Apache Karaf 2.3.0. As a result of this upgrade, the following dependencies have also been upgraded:

- Felix framework is upgraded to 4.0.3.
- OSGi Compendium is upgraded to 4.3.0.
- Pax Web is upgraded to version 1.1.9.
- Pax URL is upgraded to version 1.3.5.
- Pax Logging is upgraded to version 1.7.0.
- Jetty is upgraded to version 7.6.7.v20120910 and is now maintained by the Eclipse Foundation (`org.eclipse.jetty`).

features repository file

Since Apache Karaf 2.2.5, omitting the `name` attribute from the `features` element in a features repository file is now a deprecated syntax, which will result in a warning. In a future version of Apache Karaf, the XML schema will be changed to make the `name` attribute a required attribute.

Chapter 2. Apache ActiveMQ Issues

Fuse ESB Enterprise 7.1.0.fuse-047 uses Apache ActiveMQ 5.7.0. Since the last release, Apache ActiveMQ has been upgraded from version 5.5.1 to version 5.7.0. This introduces a few migration issues.

New Features

Java support

Apache ActiveMQ now supports Java 7 and Java 6 (the runtime is compiled with JDK 1.6 and verified against JDK 1.7).

mKahaDB persistent store

The multi-KahaDB (mKahaDB) persistent store, enables you to split message storage across multiple KahaDB store instances. For each message, the database to store the message in is chosen based on the message's *destination name* (multiple filters can be defined in order to map destination names to particular store instances).

XML element order in configuration

In versions of Apache ActiveMQ prior to 5.6.0, it was necessary to put the XML elements in a Apache ActiveMQ configuration file into alphabetical order. This restriction has been removed and it is now possible to insert sibling XML elements in any order in the configuration file.

Secure WebSocket transport

The Stomp-over-WebSocket protocol (`ws:` URL) now also supports a secure variant of the transport (`wss:` URL), which takes its SSL/TLS configuration from the `sslContext` configuration element.

Broker redelivery

In contrast to ordinary redelivery, where the redelivery logic is implemented by a consumer client, with *broker redelivery* the redelivery logic is implemented on the broker. This has the advantage that messages can be redelivered even after a specific consumer has died, but has the disadvantage that message order cannot be guaranteed. Broker redelivery can be enabled by installing the `redeliveryPlugin` on the broker.

Pluggable locking for stores

Locking on Apache ActiveMQ stores has been refactored so that the locking logic is independent of the particular store implementation. Previously, locking logic was part of a store implementation. The new architecture is more flexible, because it allows you to use any kind of locker with any kind of store database. Three locker types are currently provided: *shared file locker*, *database locker*, and *lease database locker*.

Lease database locker

The *lease database locker* is a new locker implementation that is recommended as a replacement for the default database locker. With the default database locker, it could happen that the master broker crashes without releasing the lock, so that the slave remains permanently locked out and is unable to take over from the master. This problem is solved in the lease database locker, because the master acquires only a temporary lease on the lock and must regularly renew the lease. If the master crashes, the slave can acquire the lock as soon as the lease times out.

Redelivery policy per destination

It is now possible to associate redelivery policies with specific destinations, by defining a redelivery map that associates a redelivery policy with each named destination (where the name can include wildcards).

LevelDB store

Apache ActiveMQ now supports the new LevelDB store, which uses Google's LevelDB library to maintain the indexes into the log files.

MQTT transport

Apache ActiveMQ now supports the MQTT[1] (MQ Telemetry Transport) transport. There are three variations of the MQTT transport: plain MQTT (`mqtt:` URL), MQTT+NIO (`mqtt+nio:` URL), and MQTT+NIO+SSL (`mqtt+nio+ssl:` URL).

LDAP authorization plug-in

Apache ActiveMQ now supports an LDAP authorization plug-in. You can configure the LDAP authorization plug-in by adding the `cachedLDAPAuthorizationMap` element as a child of `authorizationPlugin/map`.

Stomp 1.1

Apache ActiveMQ now supports Stomp version 1.1.

Stomp + NIO + SSL

Apache ActiveMQ now supports the Stomp+NIO+SSL protocol combination (`stomp+nio+ssl:` URL).

MS SQL JDBC driver 4.0

Apache ActiveMQ now supports using MS SQL JDBC driver 4.0 with the JDBC persistence adapter.

[1] http://mqtt.org/

API Changes

Classes removed

The following classes have been removed from Apache ActiveMQ 5.7.0:

`org.apache.activemq.thread.Valve`
: Not needed in 5.7.0. The threading model has now been refactored to use the `ThreadPoolUtils` API.

`org.apache.activemq.util.xstream.XStreamMessageTransformer`
: Replaced by `org.apache.activemq.util.oxm.XStreamMessageTransformer`.

`org.apache.activemq.transport.http.HttpTransport`

`org.apache.activemq.transport.http.HttpsTransport`

Methods removed

The following methods have been removed from Apache ActiveMQ 5.7.0:

`org.apache.activemq.console.filter.AmqMessagesQueryFilter.createConnection(java.net.URI)`
: Use `createConnection()` instead, and pass the URL to the `ConnectionFactory` when it is created.

`org.apache.activemq.broker.TransportConnector.setBrokerName`
: Use the `setBrokerService(BrokerService)` method instead.

`org.apache.activemq.command.ConsumerInfo.getSubcriptionName`
: Use the correctly spelled `getSubscriptionName` method instead.

`org.apache.activemq.command.ConsumerInfo.setSubcriptionName`
: Use the correctly spelled `setSubscriptionName` method instead.

`org.apache.activemq.transport.TransportFactory.bind(String, java.net.URI)`
: Use `bind(BrokerService, java.net.URI)` instead.

`org.apache.activemq.transport.tcp.SslTransportFactory.setKeyAndTrustManagers`

`org.apache.activemq.xbean.XBeanBrokerService.setDestroyApplicationContextOnShutdown`
: This method is not needed.

`org.apache.activemq.xbean.XBeanBrokerService.setDestroyApplicationContextOnStop`
: This method is not needed.

Deprecated classes and interfaces

The following classes and interfaces are deprecated in Apache ActiveMQ 5.7.0:

`org.apache.activemq.store.jdbc.DatabaseLocker`
: Use more the general `org.apache.activemq.broker.Locker` interface instead

`org.apache.activemq.thread.DefaultThreadPools`
: Use the `org.apache.activemq.thread.TaskRunnerFactory` class instead.

Deprecated methods

The following methods are deprecated in Apache ActiveMQ 5.7.0:

`org.apache.activemq.broker.jmx.BrokerViewMBean.getOpenWireURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerViewMBean.getSslURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerViewMBean.getStompURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerViewMBean.getStompSslURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerView.getOpenWireURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerView.getSslURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerView.getStompURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.broker.jmx.BrokerView.getStompSslURL`
: Use `BrokerViewMBean.getTransportConnectors()` or `BrokerViewMBean.getTransportConnectorByType(String)` instead.

`org.apache.activemq.selector.SimpleCharStream.getColumn`

`org.apache.activemq.store.jdbc.JDBCPersistenceAdapter.getDatabaseLocker`
: Use the `LockableServiceSupport.getLocker()` method instead.

`org.apache.activemq.thread.DefaultThreadPools.getDefaultTaskRunnerFactory`

`org.apache.activemq.selector.SimpleCharStream.getLine`

`org.apache.activemq.store.kahadb.KahaDBPersistenceAdapter.setDatabaseLockedWaitDelay(int)`
: Use `Locker.setLockAcquireSleepInterval(long)` instead.

`org.apache.activemq.store.jdbc.JDBCPersistenceAdapter.setDatabaseLocker(Locker)`
: Use the `LockableServiceSupport.setLocker(org.apache.activemq.broker.Locker)` method instead.

`org.apache.activemq.store.jdbc.JDBCPersistenceAdapter.setLockAcquireSleepInterval(long)`
: Use `Locker.setLockAcquireSleepInterval(long)` instead.

`org.apache.activemq.store.jdbc.JDBCPersistenceAdapter.setUseDatabaseLock(boolean)`
: Use `LockableServiceSupport.setUseLock(boolean)` instead.

Transport Protocol Changes

HTTPS transport

The HTTPS transport can now have its security properties configured by the `sslContext` element in the Apache ActiveMQ configuration.

Stomp 1.1

The following changes have been made to the implementation of the Stomp 1.1 protocol:

- Since Apache ActiveMQ 5.7.0, in the case where a particular header property is defined multiple times in a Stomp message, the *first* header value is now taken to be the actual value of the header. In earlier Apache ActiveMQ versions, the *last* header value is used (which does not conform with the Stomp 1.1 specification).
- Since Apache ActiveMQ 5.6.0, the spaces around header keys and values are no longer trimmed automatically (this is in order to be consistent with the Stomp 1.1 specification). This could affect the processing of headers in your Stomp clients, when migrating to Apache ActiveMQ 5.7.0.

WebSocket transport

For the WebSocket transport, it is now possible to initialize any of the parameters on the underlying the Jetty servlet holder instance by using the `websocket.` prefix on the transport server URI. This works on either the `ws:` or the `wss:` URIs.

Dependency Upgrades

Jetty and HttpClient updates

For optimum compatibility with the other components of Fuse ESB Enterprise, Apache ActiveMQ has upgraded the following dependencies:

- Jetty library is upgraded to version 7.3.1.
- HttpClient library is upgraded to version 4.1.2.

Migrating Clients

Migrating Apache ActiveMQ clients

Apache ActiveMQ clients are compatible with later versions of the broker, as long as the client version and the broker version have the same major version number. For example, if you update an Apache ActiveMQ broker to version 5.5.1, it will be compatible with clients on version 5.4.2. This makes it possible to perform an upgrade in stages, starting with the broker hosts and then followed by the client hosts.

Chapter 3. Apache Camel Issues

Summary of Apache Camel 2.9 to Apache Camel 2.10 Migration

Overview

Fuse ESB Enterprise 7.1.0.fuse-047 uses Apache Camel 2.10. Since the last release, Apache Camel has been upgraded from version 2.9 to version 2.10. This introduces a few migration issues.

Notices

The most important change in Apache Camel 2.10 is that Apache Camel now supports Java 7 (JDK 1.7).

You should also note the following important changes:

- Maven 3.0.2 or better is required to build the source.

Product dependencies

In Apache Camel 2.10, some components have had their third party dependencies upgraded. See "Product Dependencies" on page 30 for details.

API changes

In Apache Camel 2.10, the following changes have been made to the Java API:

ShutdownStrategy
: Since Apache Camel version 2.10, the `shutdownForced` and `forceShutdown` methods have been added to the `org.apache.camel.spi.ShutdownStrategy` interface.

ShutdownAware
: Since Apache Camel version 2.10, the `ShutdownAware` interface inherits from the `ShutdownPrepared` interface, which defines the additional `prepareShutdown` method.

RouteBuilder
: Since Apache Camel version 2.10, the `errorHandler` method from the `org.apache.camel.builder.RouteBuilder` interface returns `void`.

SimpleLanguage
: Since Apache Camel version 2.10, the `SimpleLanguage` constructor that takes custom start and end tokens has been removed. Use the static `SimpleLanguage.changeFunctionStartToken` method and the `SimpleLanguage.changeFunctionEndToken` method instead.

LifecycleStrategy
: Since Apache Camel version 2.10, the `onThreadPoolRemove` method and the `onErrorHandlerRemove` method have been added to the `org.apache.camel.spi.LifecycleStrategy` interface.

OnExceptionDefinition
: Since Apache Camel version 2.10, the `retryWhile(Expression)` method has been removed from the `org.apache.camel.model.OnExceptionDefinition` interface. Use the `retryWhile(Predicate)` method instead.

TypeConverter
: Since Apache Camel version 2.10, the `convertTo` methods on `org.apache.camel.TypeConverter` throw `TypeConversionException`, if an exception occurs during type conversion. New `tryConvertTo` methods have been added to `TypeConverter`, which ignore any exceptions that might occur during conversion.

Message
: Since Apache Camel version 2.10, the `getBody(type)` method and the `getHeader(name, type)` method from `org.apache.camel.Message` now throw `TypeConversionException`, if an exception occurs during type conversion.

UnitOfWork
: Since Apache Camel version 2.10, the `containsSynchronization` method has been added to the `org.apache.camel.spi.UnitOfWork` interface.

Exchange
: Since Apache Camel version 2.10, the `containsSynchronization` method has been added to the `org.apache.camel.Exchange` interface.

  Since Apache Camel version 2.10, the `isTransactionRedelivered` method has been added to the `org.apache.camel.Exchange` interface.

CamelContext
: Since Apache Camel version 2.10, the `setManagementName` method has been removed from `org.apache.camel.CamelContext`.

GenericFile
: Since Apache Camel version 2.10, the `isDirectory` method has been added to the `org.apache.camel.component.file.GenericFile` interface.

TypeConverterRegistry
: Since Apache Camel version 2.10, the `getStatistics` method has been added to the `org.apache.camel.spi.TypeConverterRegistry` interface.

GenericFileProcessStrategy
: Since Apache Camel version 2.10, the `abort` method has been added to the `org.apache.camel.component.file.GenericFileProcessStrategy` interface.

Component updates

In Apache Camel 2.10, you need to consider the following component updates:

- *Netty component*—removed the `corePoolSize` and `maxPoolSize` thread pool options.

  The API for the `ClientPipelineFactory` and `ServerPipelineFactory` abstract classes has changed.

- *CXFRS component*—the `resourceClasses` option no longer recognizes the semicolon character, `;`, as a separator for class names. Use the comma character, `,`, instead.

- *Mail component*—now excludes the dependency on the `javax.activation` JAR, because that dependency is embedded in the JVM from Java 6 onwards.
- *Test component*—Spring testing features have been moved from the `camel-test` artifact to the `camel-test-spring` artifact.
- *File component*—you can now use the `charset` option on the File component to specify the character encoding to use for reading and writing files.

Simple language

The simple language no longer trims white space surrounding an expression in the Java DSL. On the other hand, the simple language does trims white space surrounding an expression in the XML DSL by default, but you can disable this behavior by setting `trim="false"` in the `simple` element.

convertBodyTo DSL command

In this release, the `convertBodyTo` DSL command does *not* propagate the character set as an exchange property when the `charset` option is set (in previous releases, the character set was propagated in the `Exchange.CHARSET_NAME` property).

URI normalization

URI normalization now recognizes pre-existing `%nn` decimal encodings in URI strings.

Intercept

When using Apache Camel interceptors, the behavior when combining the `skipSendToOriginalEndpoint()` clause with the `when()` predicate has changed in Apache Camel 2.10.

Now, when you combine the `skipSendToOriginalEndpoint()` clause with a `when()` predicate, the original endpoint is skipped, only if the `when()` predicate evaluates to `true`. For example, in the following route, the `mock:result` endpoint will be skipped, only if the message body is equal to `Hello World`:

```
interceptSendToEndpoint("mock:foo")
    .skipSendToOriginalEndpoint()
    .when(body().isEqualTo("Hello World"))
    .to("mock:detour").transform(constant("Bye World"));

from("direct:second")
    .to("mock:bar")
    .to("mock:foo")
    .to("mock:result");
```

In previous versions, the original endpoint would have been skipped, even if the `when()` predicate evaluated to `false`.

Thread name pattern

The default pattern for thread names is now:

```
Camel (#camelId#) thread ##counter# - #name#
```

In previous releases the thread name pattern was:

```
Camel (${camelId}) thread #${counter} - ${name}
```

Poll enrich

The `pollEnrich` DSL command now blocks, if no messages are available and no timeout has been configured.

Type converters

The setting `lazyLoadTypeConverter=true` is now deprecated and will be removed in a future release. The recommended practice is to load type converters at start up time.

The Apache Camel test kit no longer lazily loads type converters.

Apache Camel now fails faster during type conversion, by throwing `TypeConversionException` to the caller.

Apache Camel no longer supports using `java.beans.PropertyEditor` for type conversion. This approach is slow, not thread safe, and uses third party JARs on the classpath, which can cause unwanted side effects.

MDC logging

The keys for MDC logging are now prefixed with `.camel`.

Wire tap

It is no longer possible to change the destination of the wire tap enterprise integration pattern from JMX.

Spring Framework

Spring version

Since Apache Camel 2.10, the version of Spring that you can use with Apache Camel *must* be 3.0.7 (or later). Spring 3.1.1 is also 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 either the 3.0 Spring schema or the 3.1 Spring schema.

The Spring 3.0 schema is located at the following Web page:

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

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-3.0.xsd">
```

Order of dependency injection in Spring 3.0

It appears that the order in which beans are dependency injected has changed in Spring 3.0. This could potentially affect your existing Apache Camel applications when you upgrade. To gain more control over the order of dependency injection, you could add the `depends-on` attribute to some of your bean definitions.

Spring 3.0 new features

For a summary of the new features in Spring 3.0, see New Features and Enhancements in Spring 3.0[1].

[1] http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/new-in-3.html

Product Dependencies

Java development kit	Since Apache Camel 2.7, the Apache Camel requires at least JDK 1.6 (the recommended version is JDK 1.6.0_18) and also supports JDK 1.7.
Maven version	Since Apache Camel 2.10, you require Maven 3.0.2 or later to build the source distribution of Apache Camel.
Apache CXF version	Since Apache Camel 2.10, the CXF component requires Apache CXF 2.6.
Jetty version	Since Apache Camel 2.10, the Jetty component is updated to Jetty 7.5.4.
Logging	Since Apache Camel 2.7, Apache Camel has switched from Apache Commons Logging to the Simple Logging Facade for Java[2] (`slf4j`). If your application code uses `log4j` logging, you must now include a dependency on the `slf4j-log4j12` artifact. For example, if you use the Maven build system, you would add the following Maven dependency to your POM file:

```
<project ...>
  <properties>
    <slf4j-version>1.6.1</slf4j-version>
    ...
  </properties>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>${slf4j-version}</version>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

[2] http://www.slf4j.org/

Where the `slf4j` version used by Apache Camel 2.8.0. is 1.6.1.

Component dependencies

Since Apache Camel 2.7, some components have had their third party dependencies upgraded, as follows:

- *HTTP4*—is upgraded to use Apache HttpClient 4.1.
- *Spring integration*—is upgraded to use the Spring integration 2.0 API.
- *Web console*—is upgraded to use Scalate 1.4.1.
- *Restlet*—is upgraded to use Restlet[3] 2.0.5.

In Apache Camel 2.8, some components have had their third party dependencies upgraded, as follows:

- *FTP*—is upgraded to use Commons Net 2.2 (from 2.0).
- *Spring Web Services*—is upgraded to release 2.0.2.
- *Cometd*—is upgraded to 2.1.0 (from 1.0.1).

In Apache Camel 2.9, some components have had their third party dependencies upgraded, as follows:

- *Cometd*—is upgraded to 2.3.1 (from 2.1.1).
- *EasyMock*—is upgraded to 3.0 (from 2.5.2).
- *EHCache*—is upgraded to 2.4.3 (from 2.3.0).
- *Jackrabbit*—is upgraded to 2.2.4 (from 1.5.5).
- *JCR API*—is upgraded to 2.0 (from 1.0).
- *OGNL*—is upgraded to 3.0.2 (from 2.7.3).
- *XStream*—is upgraded to 1.4.1 (from 1.3.1).

In Apache Camel 2.10, some components have had their third party dependencies upgraded, as follows:

- *AHC*—is upgraded to 1.7.5 (from 1.6.5).

[3] http://www.restlet.org/

- *AWS*—is upgraded to 1.3.10 (from 1.2.2).
- *commons-codec*—is upgraded to 1.6 (from 1.4).
- *commons-net*—is upgraded to 3.1.0 (from 2.2).
- *CXF*—is upgraded to 2.6.1 (from 2.5.1).
- *EHCache*—is upgraded to 2.5.1 (from 2.4.3).
- *Freemarker*—is upgraded to 2.3.19 (from 2.3.18).
- *Google App Engine*—is upgraded to 1.6.6 (from 1.5.0).
- *Groovy*—is upgraded to 1.8.6 (from 1.8.5).
- *Hadoop*—is upgraded to 1.0.3 (from 0.20.203.0).
- *HTTP4 core*—is upgraded to 4.1.4 (from 4.1.2).
- *HTTP4 client*—is upgraded to 4.1.3 (from 4.1.2).
- *Hazelcast*—is upgraded to 2.0.2 (from 1.9.4.4).
- *Hawtbuf*—is upgraded to 1.9 (from 1.7).
- *Jackson*—is upgraded to 1.9.7 (from 1.9.2).
- *Jackrabbit*—is upgraded to 2.2.11 (from 2.2.4).
- *Jasypt*—is upgraded to 1.9.0 (from 1.7).
- *Javax Mail*—is upgraded to 1.4.5 (from 1.4.4).
- *Jersey*—is upgraded to 1.12 (from 1.10).
- *JClouds*—is upgraded to 1.4.0 (from 1.3.1).
- *Jettison*—is upgraded to 1.3.1 (from 1.3).
- *Jetty*—is upgraded to 7.5.4 (from 7.5.3).
- *JRuby*—is upgraded to 1.6.7 (from 1.6.6).
- *JSCH*—is upgraded to 0.1.48 (from 0.1.44).
- *JuEL*—is upgraded to 2.1.4 (from 2.1.3).

- *Kratti*—is upgraded to 0.4.5 (from 0.4.1).
- *Logback*—is upgraded to 1.0.6 (from 1.0.0).
- *Lucene*—is upgraded to 3.6.0 (from 3.0.3).
- *MyBatis*—is upgraded to 3.1.1 (from 3.0.6).
- *Netty*—is upgraded to 3.5.1 (from 3.2.6).
- *OGNL*—is upgraded to 3.0.4 (from 3.0.2).
- *QPid*—is upgraded to 0.16 (from 0.12).
- *QuickFIX/J*—is upgraded to 1.5.2 (from 1.5.1).
- *Restlet*—is upgraded to 2.0.14 (from 2.0.10).
- *SNMP*—is upgraded to 1.10.1 (from 1.8.1).
- *Solr*—is upgraded to 3.6.0 (from 3.5.0).
- *Shiro*—is upgraded to 1.2.0 (from 1.1.0).
- *Stringtemplate*—is upgraded to 3.2.1 (from 3.0).
- *Spring*—is upgraded to 3.0.7/3.1.1 (from 3.0.6).
- *Spring integration*—is upgraded to 2.1.2 (from 2.0.5).
- *Spring security*—is upgraded to 3.1.0 (from 3.0.7).
- *Spymemcached*—is upgraded to 2.8.0 (from 2.5).
- *Tagsoup*—is upgraded to 1.2.1 (from 1.2).
- *Woodstox*—is upgraded to 4.1.2 (from 4.1.1).
- *XStream*—is upgraded to 1.4.2 (from 1.4.1).
- *XML Security*—is upgraded to 1.5.1 (from 1.4.5).

Namespace Changes

Namespace for the Spring DSL schema

The XML namespace for the Spring DSL schema has changed between Apache Camel 1.x and Apache Camel 2.2, as follows:

Old XML schema namespace:

```
http://activemq.apache.org/camel/schema/spring
```

New XML schema namespace:

```
http://camel.apache.org/schema/spring
```

Moreover, when specifying the `xsi:schemaLocation` attribute, you need to specify the location of the *new* XML schema.

Example

You need to update all of your old Spring XML configuration files to use the new schema and the new schema location. For example, the namespace settings (which are typically defined in a Spring `bean` element) should be changed as follows:

Old namespace definitions in `bean` element:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:camel="http://act
ivemq.apache.org/camel/schema/spring"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans ht
tp://www.springframework.org/schema/beans/spring-beans-2.5.xsd

         http://activemq.apache.org/camel/schema/spring ht
tp://activemq.apache.org/camel/schema/spring/camel-spring.xsd">
```

New namespace definitions in `bean` element:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans ht
tp://www.springframework.org/schema/beans/spring-beans-2.5.xsd

         http://camel.apache.org/schema/spring ht
tp://camel.apache.org/schema/spring/camel-spring.xsd">
```

JAR Dependencies

JAR dependencies

The following JAR dependencies have changed:

- From Apache Camel 2.4 onwards, the OSGi integration code and the Spring integration code have been decoupled, so that there is no longer any need for the following artifacts:
  - `camel-osgi`
  - `camel-spring-osgi`

  In fact, these artifacts are no longer provided with Apache Camel 2.4. If you want to use the integration with Spring, you only need to install the `camel-spring` artifact.
- From Apache Camel 2.5 onwards, the following test artifacts have been deprecated and will be removed in a future release:
  - `camel-core-tests`
  - `camel-spring-tests`

  In future, you should use only the `camel-test` artifact, if you want to use the Apache Camel test kit.
- From Apache Camel 2.9 onwards, the following dependencies have changed:
  - `commons-management` is no longer required as a dependency.
  - The Spring JARs are no longer needed for Apache Camel to enlist in JMX.
- Since Apache Camel 2.10, Spring testing features have been moved from the `camel-test` artifact to the `camel-test-spring` artifact.

DSL Changes

DSL changes

Table 3.1 on page 36 provides an overview of the DSL commands that have been renamed in Apache Camel 2.0.

Table 3.1. Renamed DSL Commands

Old Java DSL Name	Old Spring DSL Name	New DSL Name
`splitter`	`splitter`	`split`
`resequencer`	`resequencer`	`resequence`
`aggregator`	`aggregator`	`aggregate`
`delayer`	`delayer`	`delay`
`throttler`	`throttler`	`throttle`
`expression`	`expression`	`language`
`tryBlock`	`try`	`doTry`
`handle`	`catch`	`doCatch`
`finallyBlock`	`finally`	`doFinally`
`intercept`	`intercept`	`interceptFrom`
`thread`	`thread`	`threads`
`bean` (in `SpringBuilder` class)		`lookup`
`throwFault`		*Removed*

errorHandler changes

The following changes were made to the `errorHandler` DSL command in Apache Camel version 2.1:

- In the Java DSL the, `errorHandler` DSL command can now be configured only on a route or a Camel context. In the case of routes, you must set it directly after the `from` DSL command.

- In Spring DSL, the `errorHandlerRef` attribute is now available only on the `camelContext` and `route` elements.

adviceWith changes

Since Apache Camel version 2.3, the `adviceWith()` DSL command now takes `CamelContext` as its first argument.

onException changes

Since Apache Camel version 2.3, the Java DSL now requires `onException` (and similar clauses) to appear at the *start* of the route, otherwise Apache Camel will fail to start the route.

redeliveryPolicy changes

Since Apache Camel version 2.7, the `ref` attribute on the `redeliveryPolicy` element has been removed. Use the `redeliveryPolicyRef` attribute either on the `onException` element or on the `errorHandler` element, instead.

resequence changes

Since Apache Camel version 2.7, the `batch-config` element or the `stream-config` element, if present, must now appear as the *first* child element of the `resequence` element (in previous versions, they appeared as the last child element). For example, the `stream-config` element must now appear as follows inside a route:

```
<camelContext id="camel" xmlns="ht
tp://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:start"/>
    <resequence>
      <stream-config capacity="5000" timeout="4000"/>
      <simple>in.header.seqnum</simple>
      <to uri="mock:result" />
    </resequence>
  </route>
</camelContext>
```

sortBody removed

Since Apache Camel version 2.7, the `sortBody()` command has been removed from the Java DSL. Use `sort(body())` instead.

sort changes

Since Apache Camel version 2.7, the expression in a Spring DSL `sort` element is no longer enclosed inside an `expression` element. For example, to sort the contents of the current message body, you must now specify the `sort` element as follows in the Spring DSL:

```
<sort>
  <simple>body</simple>
</sort>
```

Whereas the old sort syntax (prior to version 2.7) would have been:

```
<sort>
  <expression>
    <simple>body</simple>
  </expression>
</sort>
```

wireTap changes

Since Apache Camel version 2.8, when using the send-new-message mode, you can now specify headers directly in the Java DSL using the `setExchangeHeader()` clause. For example:

```
from("direct:start")
    .wireTap("direct:tap")
        // create the new tap message body and headers
        .newExchangeBody(constant("Bye World"))
        .newExchangeHeader("id", constant(123))
        .newExchangeHeader("date", simple("${date:now:yyyyMM
dd}"))
    .end()
```

In the XML DSL, use the `setHeader` child element, as follows:

```
<route>
    <from uri="direct:start"/>
    <wireTap uri="direct:tap">
        <!-- create the new tap message body and headers -->
        <body><constant>Bye World</constant></body>
        <setHeader headerName="id"><constant>123</con
stant></setHeader>
       <setHeader headerName="date"><simple>${date:now:yyyyM
Mdd}</simple></setHeader>
    </wireTap>
    ...
</route>
```

Since Apache Camel 2.10, it is no longer possible to change the destination of the wire tap enterprise integration pattern from JMX.

throttle changes

Since Apache Camel version 2.8, in the XML DSL, the `maximumRequestsPerPeriod` attribute of the `throttle` element has been removed. You now specify the maximum requests per period using an expression that appears as a child of of the `throttle` element. For example, to throttle the message rate to a maximum of three messages every 10 seconds:

```
<route>
    <from uri="seda:a"/>
    <!-- throttle 3 messages per 10 sec -->
    <throttle timePeriodMillis="10000">
        <constant>3</constant>
        <to uri="mock:result"/>
    </throttle>
</route>
```

The advantage of using an expression here is that it enables you to specify the rate dynamically at run time.

bean, marshal, and unmarshal elements

Since Apache Camel version 2.8, in the XML DSL, the `bean`, `marshal`, and `unmarshal` elements are no longer permitted to have any child elements.

idempotentConsumer changes

Since Apache Camel version 2.8, the JDBC and JPA-based idempotent repositories now contain a `createdAt` column.

Removed clauses that use an expression builder

Since Apache Camel version 2.8, the following DSL methods, which used an `ExpressionClause` to build up an expression, have been removed:

```
aggregate(...).completionPredicate()
catch(...).onWhen()
idempotentConsumer(...).expression()
onCompletion(...).onWhen()
onException(...).onWhen()
idempotentConsumer()
split(...).expression()
try(...).onWhen()
expression() on WhenDefinition
```

Instead of building up the expression using an expression clause builder, you can use the form of DSL method that specifies the expression directly as an argument. For example:

```
aggregate(...).completionPredicate(body().isEqualTo("END"))...
```

convertBodyTo changes

Since Apache Camel 2.10, the `convertBodyTo` DSL command does *not* propagate the character set as an exchange property when the `charset` option is set (in previous releases, the character set was propagated in the `Exchange.CHARSET_NAME` property).

API Changes

API changes

The following changes have been made to the Java API:

ProducerTemplate
: Since Apache Camel 2.0, the `org.apache.camel.ProducerTemplate` class has been refactored so that `sendBody` methods now return void for *InOnly* messaging. Use one of the `requestBody` methods for *InOut* messaging. See "Producer and Consumer Templates" in *Programming EIP Components* for more details.

Exchange
: Since Apache Camel 2.0, all specializations of `org.apache.camel.Exchange` are now removed. You should now always use the `org.apache.camel.impl.DefaultExchange` type. After analyzing the how exchanges are used, it was realized that the specialized exchanges were not really necessary and by removing them we can avoid a lot of unnecessary copying and improve throughput.

  One implication of this change is that `Producer` and `Consumer` types no longer use generic type declarations. For example, instead of referring to `Producer<DefaultExchange>` you can just refer to a plain `Producer`.

  Since Apache Camel version 2.10, the `containsSynchronization` method has been added to the `org.apache.camel.Exchange` interface.

  Since Apache Camel version 2.10, the `isTransactionRedelivered` method has been added to the `org.apache.camel.Exchange` interface.

Exchange `getFault()` and `setFault()` methods
: Since Apache Camel 2.0, the `getFault()` and `setFault()` methods are now removed from `Exchange`. Faults represent application specific errors and are recognized by some protocols. Consequently, it makes more sense to store a fault as the *Out* message in an exchange (accessed using `getOut()` and `setOut()`). The `org.apache.camel.Message` interface now exposes the boolean `isFault()` and `setFault()` methods that are used to identify *Out* messages that represent faults.

  Because faults represent persistent errors (as opposed to exceptions, which represent transient errors), Camel does not try (as in previous versions) to recover from them (for example, the error handler does not trigger) unless handling faults as exceptions is explicitly enabled.

AggregationStrategy
: Since Apache Camel 2.0, the following changes have been made to the implementation of `org.apache.camel.processor.aggregate.AggregationStrategy`:

  - The `aggregate()` method is now invoked on the very first exchange. For this first invocation, the `oldExchange` parameter is `null`.
  - The payload is now always stored in the *In* message when you do custom aggregation using this strategy interface.

Aggregator
: Since Apache Camel 2.3, the aggregator has been re-implemented and some options have been replaced. In particular, the algorithms for determining batch completeness have changed significantly, so that the `batchSize`, `outBatchSize`, `batchTimeout`, and `batchConsumer` options are no longer supported. To understand the new mechanisms for completeness testing, it is recommended that you read "Aggregator" in *Implementing Enterprise Integration Patterns*.

CamelContext
: Since Apache Camel version 2.1, the following changes have been made to the `CamelContext` class:

  - The `shouldStartContext()` method is replaced by the `autoStartup()` method.
  - The `getLifecycleStrategy()` method has been renamed `getLifecycleStrategies()` and now returns a `java.util.List`.

  Since Apache Camel version 2.5, the following changes have been made to the `CamelContext` class:

  - The `stopRoute()` method is now integrated with the graceful shutdown strategy (see "Controlling Start-Up and Shutdown of Routes" in *Implementing Enterprise Integration Patterns*). You can revert to the old behavior by specifying an explicit timeout.

Since Apache Camel version 2.10, the `setManagementName` method has been removed from `org.apache.camel.CamelContext`.

ManagementNamingStrategy
: Since Apache Camel version 2.1, the `org.apache.camel.spi.ManagementNamingStrategy` has had methods renamed and method signatures changed in order to accomodate the JMX features in this release.

PollingConsumerPollStrategy
: Since Apache Camel version 2.3, the `begin()` method from `org.apache.camel.spi.PollingConsumerPollStrategy` returns a boolean value, where `true` indicates that polling can now start, while `false` indicates that polling should be skipped.

RoutePolicy
: Since Apache Camel version 2.3, the `org.apache.camel.spi.RoutePolicy` interface has the new method, `onInit()`.

  Since Apache Camel version 2.9, the `org.apache.camel.spi.RoutePolicy` interface has the following new methods: `onRemove`, `onStart`, `onStop`, `onSuspend`, and `onResume`.

Message
: Since Apache Camel version 2.3, the `org.apache.camel.Message` interface has the new method, `removeHeaders()`.

  Since Apache Camel version 2.10, the `getBody(type)` method and the `getHeader(name, type)` method from `org.apache.camel.Message` now throw `TypeConversionException`, if an exception occurs during type conversion.

DefaultComponent and DefaultEndpoint
: Since Apache Camel version 2.3, the `getExecutorService()` and `setExecutorService()` methods have been removed from the `org.apache.camel.impl.DefaultComponent` and `org.apache.camel.impl.DefaultEndpoint` classes. To create a thread pool, use the `ExecutorServiceManager` object that is returned by the `CamelContext.getExecutorServiceManager()` method.

For full details of the new threading model, see "Threading Model" in *Implementing Enterprise Integration Patterns*.

If you have developed a custom component and you have implemented an endpoint by inheriting from the `DefaultEndpoint` class, it is strongly recommended that you override the `doStart()` and `doStop()` methods, instead of overriding the `start()` and `stop()` methods. Since Apache Camel version 2.7, the default endpoint implementation ensures that the `doStart()` and `doStop()` methods are called once and once only.

GenericFile
: Since Apache Camel version 2.3, the `org.apache.camel.component.file.GenericFile` class is no longer serializable (does not inherit from `java.io.Serializable`).

  Since Apache Camel version 2.10, the `isDirectory` method has been added to the `org.apache.camel.component.file.GenericFile` interface.

RouteDefinition
: Since Apache Camel version 2.3, the `adviceWith()` method from `org.apache.camel.model.RouteDefinition` takes a `CamelContext` instance as its first parameter.

toAsync
: Since Apache Camel version 2.4, the `toAsync()` DSL command has been removed. Asynchronous dispatch is now implemented directly (where appropriate) in specific Apache Camel components and DSL commands.

Policy
: Since Apache Camel version 2.4, the `org.apache.camel.spi.Policy` interface has the new method, `beforeWrap()`. For quick migration of your `Policy` classes, simply add an empty method implementation.

onException
: Since Apache Camel version 2.4, the `retryUntil` option on `onException` has been renamed to `retryWhile`, because this reflects the meaning of the option more accurately (it continues to retry while its argument is true).

Routing Slip

Since Apache Camel version 2.4, you can use either a string or an expression to specify the name of the routing slip header. The Spring DSL has changed, such that the `headerName` attribute is now replaced by the `headerName` child element. For example, to specify the name of the routing slip header to be `myHeader`, use an XML fragment like the following:

```
<route>
  <from uri="direct:a"/>
  <routingSlip ignoreInvalidEndpoints="true">
    <headerName>myHeader</headerName>
  </routingSlip>
</route>
```

ProducerTemplate

Since Apache Camel version 2.4, all `sendBody` and `requestBody` methods from the `ProducerTemplate` class throw a `CamelExecutionException`, which wraps the original exception.

RouteBuilder

Since Apache Camel version 2.4, the `simple` and `xpath` expression builder methods are built into the `RouteBuilder` class. It is, therefore, no longer necessary to use static imports to access these languages. In your existing code, the Java compiler might complain, if you use static imports for `simple` and `xpath`. To fix this, just remove the static imports.

Since Apache Camel version 2.10, the `errorHandler` method from the `org.apache.camel.builder.RouteBuilder` interface returns `void`.

ManagementAware

Since Apache Camel version 2.6, the `org.apache.camel.spi.ManagementAware` interface is deprecated. If you want to expose a bean through JMX, use the Spring JMX annotations instead.

For example, you would annotate a class using the Spring `@ManagedResource` annotation, and then annotate each managed attribute using `@ManagedAttribute` and each managed operation using `@ManagedOperation`—see the Spring Reference Guide[4] for details.

[4] http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jmx.html#jmx-interface-metadata

DataFormat
: Since Apache Camel version 2.6, when implementing a custom data format class, you can also optionally implement the `Service` and `CamelContextAware` interfaces in order to receive callbacks at initialization and shutdown time. For example, you can implement the `MyCustomDataFormat` class with callbacks, as follows:

```
// Java
...
public class MyCustomDataFormat implements DataFormat,
Service, CamelContextAware {
    ...
}
```

ScheduledPollConsumer
: Since Apache Camel version 2.6, the `poll` method in `ScheduledPollConsumer` now returns the number of messages actually processed from the poll.

LoggingLevel
: Since Apache Camel version 2.7, the `FATAL` logging level has been removed from `org.apache.camel.LoggingLevel`. Use `ERROR` instead.

BinaryPredicate
: Since Apache Camel version 2.7, the `getLeftValue()` and `getRightValue()` methods have been removed from `org.apache.camel.BinaryPredicate`.

PropertiesParser
: Since Apache Camel version 2.7, the `parsePropertyValue()` method from the `org.apache.camel.component.properties.PropertiesParser` interface has been renamed to `parseProperty()` and the signature has changed from:

```
// Java
// Versions prior to 2.7
String parsePropertyValue(String value);
```

To the new signature:

```
// Java
// Camel 2.7 or later
String parseProperty(String key, String value, Properties
 properties);
```

ManagedCamelContext

Since Apache Camel version 2.7, the `sendBody` and `requestBody` methods from the `org.apache.camel.management.mbean.ManagedCamelContext` take an `Object` instead of a `String` as their second parameter. Two new methods, `sendStringBody` and `requestStringBody`, have been added that take a `String` as their second parameter.

The new method signatures are as follows:

```
// NEW method signagures
void sendBody(String endpointUri, Object body);
Object requestBody(String endpointUri, Object body);
void sendStringBody(String endpointUri, String body);
Object requestStringBody(String endpointUri, String body);
```

errorHandler

Since Apache Camel version 2.8, the deprecated `handled` methods on `errorHandler` have been removed. If you need the functionality of the handled method (for example, to set `handled(false)`), you must migrate to use `onException`, which still supports the `handled` method.

InterceptStrategy

Since Apache Camel version 2.8, the semantics of the `wrapProcessorInInterceptors` method from the `org.apache.camel.spi.InterceptStrategy` class have changed, so that the actual `OutputDefinition` instance is passed to `wrapProcessorInInterceptors`.

ExecutorServiceStrategy

Since Apache Camel version 2.9, the `org.apache.camel.spi.ExecutorServiceStrategy` is deprecated in favor of the new `ExecutorServiceManager` class, which has a slightly reduced and improved API. For more details, see "Threading Model" in *Implementing Enterprise Integration Patterns*.

ResourceBasedComponent
: Since Apache Camel version 2.9, the `org.apache.camel.component.ResourceBasedComponent` class from `camel-spring` is deprecated. Use `org.apache.camel.impl.DefaultComponent` from `camel-core` instead.

ResourceBasedEndpoint
: Since Apache Camel version 2.9, the `org.apache.camel.component.ResourceBasedEndpoint` class from `camel-spring` is deprecated in favor of the new `org.apache.camel.component.ResourceEndpoint` in `camel-core`, and its dependency on Spring JARs has been removed. Use the new APIs on `ResourceEndpoint` to load resources.

PollingConsumerPollingStrategy
: Since Apache Camel version 2.9, the signature of the method, `beforePoll`, from the `org.apache.camel.PollingConsumerPollingStrategy` class has changed.

ServiceSupport
: Since Apache Camel version 2.9, `ServiceSupport` does not handle child services any more. The new class, `org.apache.camel.support.ChildServiceSupport`, now handles this case. Additionally the `org.apache.camel.StatefulService` interface has been added, which provides access to `ServiceSupport` methods without knowing the implementation.

DataFormatResolver
: Since Apache Camel version 2.9, `DataFormatResolver` does not support the `resolveDataFormatDefinition` method anymore. The current implementations for OSGi and Default were exactly the same, so this code was moved to `DefaultCamelContext`.

CamelContext
: Since Apache Camel version 2.9, all methods in `CamelContext` that reference model elements are deprecated. The new interface,

`org.apache.camel.model.ModelCamelContext`, has been created to hold these methods

FileUtil
: Since Apache Camel version 2.9, `renameFile` from `org.apache.camel.util.FileUtil` has a new `copyAndDeleteOnRenameFail` boolean option and throws `IOException` in case of an exception.

CamelTestSupport
: Since Apache Camel version 2.9, the context, template, and consumer fields in `orb.apache.camel.test.CamelTestSupport` are no longer static.

WrappedFile
: Since Apache Camel version 2.9, the `org.apache.camel.WrappedFile` interface has been introduced, as an abstraction of the `GenericFile` class, for use outside the file component.

ScriptBuilder
: Since Apache Camel version 2.9, all Spring-related methods in the `org.apache.camel.builder.script.ScriptBuilder` class have been removed (for example, methods taking `org.springframework.core.io.Resource` as an argument).

ErrorHandlerBuilder
: Since Apache Camel version 2.9, the `org.apache.camel.builder.ErrorHandlerBuilder` interface's `configure` method takes a `org.apache.camel.spi.RouteContext` argument.

ShutdownStrategy
: Since Apache Camel version 2.10, the `shutdownForced` and `forceShutdown` methods have been added to the `org.apache.camel.spi.ShutdownStrategy` interface.

ShutdownAware
: Since Apache Camel version 2.10, the `ShutdownAware` interface inherits from the `ShutdownPrepared` interface, which defines the additional `prepareShutdown` method.

SimpleLanguage
: Since Apache Camel version 2.10, the `SimpleLanguage` constructor that takes custom start and end tokens has been removed. Use the static `SimpleLanguage.changeFunctionStartToken` method and the `SimpleLanguage.changeFunctionEndToken` method instead.

LifecycleStrategy
: Since Apache Camel version 2.10, the `onThreadPoolRemove` method and the `onErrorHandlerRemove` method have been added to the `org.apache.camel.spi.LifecycleStrategy` interface.

OnExceptionDefinition
: Since Apache Camel version 2.10, the `retryWhile(Expression)` method has been removed from the `org.apache.camel.model.OnExceptionDefinition` interface. Use the `retryWhile(Predicate)` method instead.

TypeConverter
: Since Apache Camel version 2.10, the `convertTo` methods on `org.apache.camel.TypeConverter` throw `TypeConversionException`, if an exception occurs during type conversion. New `tryConvertTo` methods have been added to `TypeConverter`, which ignore any exceptions that might occur during conversion.

UnitOfWork
: Since Apache Camel version 2.10, the `containsSynchronization` method has been added to the `org.apache.camel.spi.UnitOfWork` interface.

TypeConverterRegistry
: Since Apache Camel version 2.10, the `getStatistics` method has been added to the `org.apache.camel.spi.TypeConverterRegistry` interface.

GenericFileProcessStrategy
: Since Apache Camel version 2.10, the `abort` method has been added to the `org.apache.camel.component.file.GenericFileProcessStrategy` interface.

Removed classes

The following classes have been removed from the API:

```
org.apache.camel.processor.CompositeProcessor
org.apache.camel.impl.ProducerTemplateProcessor
org.apache.camel.impl.NoPolicy
org.apache.camel.spi.Provider
org.apache.camel.spring.handler.LazyLoadingBeanDefinitionParser

org.apache.camel.spring.handler.ScriptDefinitionParser
org.apache.camel.spring.remoting.SendBeforeInterceptor
org.apache.camel.spring.spi.SpringConverters
```

Since Apache Camel version 2.8, the following classes have been removed:

```
org.apache.camel.spi.ScriptEngineResolver
```

Moved classes

The following classes have moved to a different Java package:

- `PollingConsumerPollStrategy` has moved from `org.apache.camel` to `org.apache.camel.spi`.
- `PatternBasedPackageScanFilter` has moved from `org.apache.camel.impl.scan` to `org.apache.camel.spring`.

The following classes have been renamed:

- `org.apache.camel.management.event.ExchangeFailureEvent` has been renamed to `ExchangeFailedEvent`.

Since Apache Camel version 2.7, the following classes have been renamed:

- `org.apache.camel.processor.Logger` has been renamed to `org.apache.camel.processor.CamelLogger` (and now uses `slf4j` as the logger).

Since Apache Camel version 2.7, the following classes have moved to a different package:

- `JdbcAggregationRepository` has moved from `org.apache.camel.component.jdbc.aggregationrepository` to `org.apache.camel.processor.aggregate.jdbc`.

Since Apache Camel version 2.8, the following classes have moved to a different package:

- `GZIPHelper` has moved from the HTTP, HTTP4, and GHttp components to the `org.apache.camel.util` package in `camel-core`.
- `CxfHeaderFilterStrategy` has moved from `org.apache.camel.component.cxf` (in `camel-cxf`) to `org.apache.camel.component.cxf.common.header` (in `camel-cxf-transport`).
- `ProxyInstantiationException` has moved from `org.apache.camel.impl` to `org.apache.camel`.

Since Apache Camel version 2.9, the following classes have moved to a different package:

- `Ordered` has moved from `org.apache.camel.util` to `org.apache.camel`.
- `BytesSource` and `StringSource` have moved from `org.apache.camel.converter.jaxp` to `org.apache.camel`.
- `TimeoutMap` has moved from `org.apache.camel.util` to `org.apache.camel`.

- `DefaultTimeoutMap` has moved from `org.apache.camel.util` to `org.apache.camel.support`.
- Some management interfaces have moved from `org.apache.camel.management` to `org.apache.camel.spi.management`.
- `DefaultChannel` has moved from `org.apache.camel.processor` to `org.apache.camel.processor.interceptor`.
- `ModelHelper` has moved from `org.apache.camel.util` to `org.apache.camel.model`.
- `ServiceSupport` has moved from `org.apache.camel.impl` to `org.apache.camel.support` (the old class is kept, but is deprecated).
- `EventNotifierSupport` has moved from `org.apache.came.management` to `org.apache.camel.support` (the old class is kept, but is deprecated).

Component Updates

File and FTP components

Since Apache Camel version 2.0, the following changes have been made to the File and the FTP components:

- Only a directory name can be specified directly in the endpoint, *not a filename*. In other words, you must specify a File endpoint with the syntax, `file:`*`directoryName`*`[?`*`options`*`]` and an FTP component with the syntax, `ftp://[`*`username`*`@]`*`hostname`*`[:`*`port`*`]/`*`directoryname`*`[?`*`options`*`]`. It is still possible to select individual files, however, by appending the `fileName` option, which enables you to specify files using the file expression language[5].
- File producer endpoints and FTP producer endpoints now overwrite existing files by default (previously, the default behavior was to append to files).You can use the new `fileExist` option to specify what happens when a producer attempts to write a file that already exists. Valid values for the `fileExist` option are: `Override`, `Append`, `Fail`, or `Ignore`.

Important

Both the File component and the FTP component have been extensively rewritten in Apache Camel 2.0. In particular, many of the component options have been renamed or modified. It is recommended that you check the configuration of your File and FTP endpoints against the latest component documentation—see File2 in *EIP Component Reference* and FTP2 in *EIP Component Reference*.

Since Apache Camel 2.8, the default value of the `useFixedDelay` option has changed from `false` to `true`.

Since Apache Camel 2.10, you can use the `charset` option on the File component to specify the character encoding to use for reading and writing files.

JMS component

The JMS `correlationId` is determined differently to before. Now, JMS always uses the `messageId` as the `correlationId`, if configured to do so by setting `useMessageIDAsCorrelationID` to `true`. If the setting is `false`,

[5] http://camel.apache.org/file-language.html

the `JMSCorrelationID` header value is used, if present, and the `messageId` value is used as a fallback, if the header is not present.

Since Apache Camel version 2.5, the following changes have been made to the JMS component:

- The JMS 1.0.2 API is no longer supported.
- Durable topic subscribers now must provide a `clientId` value, otherwise Apache Camel fails fast on start-up.

Since Apache Camel version 2.6, a URI of the form, `jms:queue:Foo?replyTo=FooReply&preserveMessageQos=true`, can be used to set the `JMSReplyTo` header in the outgoing message, even if the `Exchange` has an *InOnly* MEP. In previous Apache Camel versions, the URI `replyTo` option was ignored for *InOnly* exchanges. For more details, see JmsProducer in *EIP Component Reference*.

Since Apache Camel version 2.8, the default cache level has changed for JMS endpoints, from `CACHE_CONSUMER` to `CACHE_AUTO`. When transactions are enabled, `CACHE_AUTO` resolves to `CACHE_NONE` (caching disabled), in order to guarantee compatibility with XA transactions. If you do *not* use XA transactions, however, it is *strongly* recommended that you set the caching level explicitly to `CACHE_CONSUMER` to improve performance. For more details, see Configuring the JMS Component in *EIP Transaction Guide*.

List component

The List component has been renamed to Browse.

Bean component

The Bean component now enforces stricter matching of bean method parameters. Previously, if a parameter could not be converted to the appropriate type, it was passed as `null`. Now, all parameters must be convertible to the relevant types.

Direct component

The `allowMultipleConsumers` option has been removed. A Direct endpoint can now have only *one* consumer.

HTTP component

Since Apache Camel version 2.2, HTTP proxy configuration is set using `CamelContext` properties instead of Java system properties.

Since Apache Camel version 2.3, the HTTP authentication options have been renamed, to avoid clashing with the `username` and `password` URI parameters. Authentication is now set using the `authUsername`, `authPassword`, and `authDomain` options. In addition, you must specify the authentication method using the `authMethod` option, which can take the values `Basic`, `Digest`, or `NTLM`.

For the full list of HTTP authentication options, see HTTP in *EIP Component Reference*.

SEDA and VM components

Since Apache Camel 2.0, these components support request/reply semantics. Endpoints of SEDA or VM type will wait for a reply, if a reply is expected.

Since Apache Camel 2.3, the size of the SEDA queue (which holds incoming exchanges) is unbounded, whereas previously it had a maximum size of 1000.

MINA component

Since Apache Camel version 2.3, the header key, `MinaConsumer.HEADER_CLOSE_SESSION_WHEN_COMPLETE`, is replaced by `MinaConstants.MINA_CLOSE_SESSION_WHEN_COMPLETE`.

Jetty component

Since Apache Camel version 2.3, the Jetty component has been upgraded from Jetty 6.1.22 to Jetty 7.0.1. The Jetty API has changed significantly between these two versions. Because Jetty is now hosted at Eclipse[6], all Java packages have been renamed. If you use the Jetty API directly, you could use the Jetty 6 to Jetty 7 migration tool[7] from Eclipse to simplify the migration of your source code.

CXF component

Since Apache Camel version 2.3, the CXF component's `PAYLOAD` mode has been improved to delegate all SOAP message parsing to CXF.

Since Apache Camel version 2.7, when using POJO data format, the CXF message attachments are *not* copied to the attachment properties of the *In* message—which means it is no longer possible to access a CXF attachment by calling `org.apache.camel.Message.getAttachment()`. But the attachments are still accessible from the message body. This avoids the problem affecting earlier versions of Apache Camel, where the attachments must be explicitly cleared on the message, in order to avoid being copied into the CXF response.

Since Apache Camel version 2.8, the Camel transport for CXF and the `cxfbean` component have been moved from the `camel-cxf` artifact to the

[6] http://www.eclipse.org
[7] http://wiki.eclipse.org/Jetty/Howto/Upgrade_from_Jetty_6_to_Jetty_7

camel-cxf-transport artifact. If you need to use either of these components, you must a Maven dependency on the camel-cxf-transport artifact.

FTP component

Since Apache Camel version 2.4, the following changes have been made to the FTP component:

- The ftps default port has been changed from 2222 to 21.
- The ftps protocol now uses a secure data channel while transferring files, whereas previously only secure login was enabled. You can now use the disableSecureDataChannelDefaults, execProt, and execPbsz options to customize the behavior of the security channel. See FTP2 in *EIP Component Reference* for details.
- The FTP base directory can now be specified using an absolute path. In the endpoint URI, insert two leading forward slashes, //, to specify an absolute path, as in ftp:admin@someserver//absolutePath/foo/bar?password=secret.
- The FTP component now uses a 10 second default connect timeout (for all protocols) and a 30 second data timeout (for the ftp and ftps protocols only).

Since Apache Camel version 2.5, the following changes have been made to the FTP component:

- FTP consumer endpoints now traverse the file structure in a different way. It is recommended that you test any applications with FTP consumer endpoints to ensure that they are not affected by this change.

Netty component

Since Apache Camel 2.5, the timeout option on the Netty component has been removed, because it did not work properly.

Since Apache Camel 2.10, the corePoolSize and maxPoolSize thread pool options have been removed.

Since Apache Camel 2.10, the API for the `ClientPipelineFactory` and `ServerPipelineFactory` abstract classes has changed.

Quartz component

Since Apache Camel 2.5, if you are using the Quartz component with jobs persisted in a database, you should note that Apache Camel now resolves job names based on the endpoint URI *without* parameters. This makes it possible to change cron parameters on the same job (that is, to reschedule the job).

Since Apache Camel 2.6, the Quartz component enforces that the trigger group/trigger name combination is unique within a given component instance and throws an exception, if a name clash is detected. This does not apply to clustered quartz, however.

Printer component

Since Apache Camel 2.6, the `mediaSize` option on the Printer component uses exactly the same constants as the Java printer API; that is, underscores are used instead of hyphens in the constants.

JPA component

Since Apache Camel 2.6, when using the JPA component to store trace information generated by the Apache Camel trace interceptor, the `JpaTraceEventMessage` class now attaches the `@Lob` annotation to any fields that could contain a lot of data, such as a message body. This ensures that these large data fields are persisted as CLOB/BLOB JDBC fields (and thus avoids truncating the data).

Since Apache Camel 2.9, the JPA component has been upgraded from using the JPA1 specification to use JPA2 specification.

Servlet component

Since Apache Camel 2.7, the Servlet component does not automatically start up a Spring context any more. Hence, if you define your routes using a Spring configuration file, you must start up a Spring context explicitly. For example, if you are deploying a Apache Camel application as a Web application (`.war` file), the standard approach to starting a Spring context is to create a `ContextLoaderListener` instance in the `web.xml` file, as follows:

```
<web-app ...>
    <display-name>My Web Application</display-name>

    <!-- location of spring xml files -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
       <param-value>classpath:camel-config.xml</param-value>
    </context-param>

    <!-- the listener that kick-starts Spring -->
    <listener>
        <listener-class>org.springframework.web.context.Con
textLoaderListener</listener-class>
    </listener>

    <!-- Camel servlet -->
    ...
</web-app>
```

Where `camel-config.xml` is the Spring configuration file located on the classpath. For full details of how to deploy such a servlet application, see the `examples/camel-example-servlet-tomcat` demonstration.

JDBC aggregator component

Since Apache Camel 2.7, the JDBC aggregator component has been incorporated into the SQL component. See Using the JDBC based aggregation repository in *EIP Component Reference* for details.

Cache component

Since Apache Camel 2.8, the header names used by the cache component have changed, so that they are now prefixed by `CamelCache`. The headers are now: `CamelCacheGet`, `CamelCacheCheck`, `CamelCacheAdd`, `CamelCacheUpdate`, `CamelCacheDelete`, and `CamelCacheDeleteAll`.

The headers are now removed from the exchange, after the caching has been performed.

HTTP4 component

Since Apache Camel 2.8, the options for configuring HTTP Basic Authentication and HTTP Proxies have different names. See HTTP4 in *EIP Component Reference* for details.

Log component

The Log in *EIP Component Reference* component no longer shows stream message bodies by default, but you can use the new `showStreams` option to enable that behavior. Likewise the Log component does not automatically convert the payload to the `StreamCache` type. For that you need to explicitly enable stream caching on the route or on the Camel context.

Bean component

Since Apache Camel 2.8, the `CamelBeanMethodName` header is removed after an exchange passes through a Bean endpoint.

CSV component

Since Apache Camel 2.9, the CSV component always returns the `List<List>` type, even if only one row is returned, in order to be more consistent.

Validation component

Since Apache Camel 2.9, the Validation component has been moved from `camel-spring` to `camel-core` as it no longer depends on Spring JARs.

XSLT component

Since Apache Camel 2.9, the XSLT component has been moved from `camel-spring` to `camel-core` as it no longer depends on Spring JARs.

Mail component

Since Apache Camel 2.9, the Mail component no longer depends on the Spring APIs. Any custom `JavaMailSender` class must now implement `org.apache.camel.component.mail.JavaMailSender` instead of the corresponding Spring interface.

Since Apache Camel 2.10, the Mail component excludes the dependency on the `javax.activation` JAR, because that dependency is embedded in the JVM from Java 6 onwards.

QuickFix component

Since Apache Camel 2.9, the QuickFix component no longer depends on the Spring APIs. If you want to use the XML DSL to set up QuickFix, you must now use the `QuickfixjConfiguration` class in place of the `QuickfixjSettingsFactory` class. For more details, see Quickfix in *Apache Camel Documentation*.

CXFRS component

Since Apache Camel 2.10, the `resourceClasses` option no longer recognizes the semicolon character, `;`, as a separator for class names. Use the comma character, `,`, instead.

Test component

Since Apache Camel 2.10, Spring testing features have been moved from the `camel-test` artifact to the `camel-test-spring` artifact.

Miscellaneous Changes

Error handling

Apache Camel 2.2 uses the new `DefaultErrorHandler`, instead of the `DeadLetterChannel` (versions 1.x), as the default error handler. This new `DefaultErrorHandler` implementation does *not* catch and handle thrown exceptions, which means any exception thrown will be propagated back to the caller.

Since Apache Camel 2.5, `onException` clauses defined at `RouteBuilder` level *must* be defined before the routes, otherwise Apache Camel throws an exception while starting the routes.

Since Apache Camel 2.8, exceptions thrown while handling other exceptions in `onException` are now caught by a fallback error handler, which logs the second exception and propagates the second exception in the Exchange. The Exchange will then cease processing and fail immediately.

Stream caching

Stream caching is a feature that enables you to re-read a message body consisting of a stream type. The interaction between Apache ServiceMix and Apache Camel has now been modified in such a way that Apache Camel does *not* need to use stream caching. It is therefore disabled by default.

Note

If necessary, you can re-enable caching by adding the `streamCaching()` command to a route in the Java DSL or by setting the `route` element's `streamCaching` attribute to `true` in the Spring XML DSL.

Annotations

The following changes have been made to the Java annotations:

- The `name` attribute in `@EndpointInject` has been changed to `ref`, in order to be consistent with the other annotations (where the `@EndpointInject` annotation references an Endpoint that gets looked up in the Registry).

- Since Apache Camel 2.3, the spelling of the `parallelProcessing` attribute in `@RecipientList` has been fixed.

Case-insensitivity of header look-up

Since Apache Camel 2.0, header look-up in `org.apache.camel.Message` is case-insensitive. This means that if you look up a header using differently cased keys (such as `Foo` and `foo`), you will obtain a reference to the *same* header entry. This makes header look-up less error-prone when used with protocols such as HTTP, where header names are inherently case insensitive.

Round robin load balancing

The round robin load balancing behavior has changed in Apache Camel 2.2. In version 1.x, the round robin load balancer would fail over, if an endpoint failed to process the message. In version 2.2, the round robin load balancer does not fail over immediately, but tries to redeliver the message to the same endpoint, according to the error handling configuration.

Note

Apache Camel 2.3 will introduce an option that allows you to specify which behavior you want the endpoint to exhibit.

Splitter

Since Apache Camel 2.3, if the exchange that enters the splitter has the *InOut* message-exchange pattern (that is, a reply is expected), the splitter returns a copy of the original input message as the reply message in the *Out* message slot. To revert to the old (pre 2.3) behavior, configure the splitter to use the `UseLatestAggregationStrategy` aggregation strategy.

xpath language and XML conversions

Since Apache Camel 2.3, the `NodeList` to `String` converter has been improved to include XML tags and attributes. For example, the conversion can now produce strings like `<foo id="123">bar<year>2015</year></foo>`.

This change will affect code that relied on the old string conversion. For example, if you previously extracted strings from XML using `@XPath` annotations as in the following method signature:

```
// Java
// Camel version 2.2 or earlier:
public void credit(
        @XPath("/transaction/transfer/receiver") String name,
        @XPath("/transaction/transfer/amount") String amount
        )
```

You must now explicitly specify the `text()` node, in order to avoid including the element tags in the string, as follows:

```
// Java
// Camel version 2.3 or later
public void credit(
       @XPath("/transaction/transfer/receiver/text()") String
 name,
        @XPath("/transaction/transfer/amount/text()") String
 amount
        )
```

Converter character set

The character set used by Apache Camel converters can be set using the `org.apache.camel.default.charset` Java system property. The default is `UTF-8`.

CamelContext

Since Apache Camel 2.5, the following changes affect `CamelContext`:

- Apache Camel fails to start up, if more than one `CamelContext` instance with the same ID is registered in JMX.
- If you do not specify an ID value for a `camelContext` element, Apache Camel now assigns an automatically generated ID from the sequence, `camel-1`, `camel-2`, and so on. In previous Apache Camel versions, the ID value defaulted to the string, `camelContext`.

UuidGenerator

Since Apache Camel 2.5, Apache Camel supports the use of third party UUID generators. The default UUID generator is based on the same implementation used in Apache ActiveMQ, `ActiveMQUuidGenerator`. Some of the APIs required by this implementation (for example, the Google app engine) might not be accessible in all deployment contexts, in which case you should switch to another of the UUID generator implementations. The following implementations are provided:

- `org.apache.camel.impl.JavaUuidGenerator`.

- `org.apache.camel.impl.SimpleUuidGenerator`.
- `org.apache.camel.impl.ActiveMQUuidGenerator`.

You can set the UUID generator in either of the following ways:

- In the Java DSL, call the `CamelContext.setUuidGenerator()` method as follows:

  ```
  // Java
  getContext().setUuidGenerator(new MyCustomUuidGenerator());
  ```

- In the Spring DSL, simply instantiate a bean of the requisite type. For example, to enable the `JavaUuidGenerator`:

  ```
  <bean id="javaUuidGenerator"
        class="org.apache.camel.impl.JavaUuidGenerator" />
  ```

JMX fails fast on start-up

Since Apache Camel 2.5, if there is a problem with starting the JMX service, Apache Camel will not start at all. In previous versions, if JMX failed, a warning was logged (at `WARN` level) and Apache Camel continued to start up without the JMX service. The problem with the old behavior is that it could lead to errors later on—for example, some J2EE servers might throw a security exception, if you try to access a JMX MBean server at run time when JMX has failed to start.

Note

Programmers who want to exploit the fail safe mechanism for custom features now have the option of raising the `VetoCamelContextStartException` exception, whenever the `onContextStart()` method is called back on the `LifecycleStrategy` interface.

Aggregator

Since Apache Camel 2.5, by default, the aggregator pattern uses a synchronous invocation to processes completed aggregated exchanges. This ensures that no internal task queue is used, effectively throttling the incoming stream of messages. To switch to asynchronous operation, enable the parallel processing option.

Maven archetypes

Since Apache Camel 2.7, the `camel-archetype-war` Maven archetype is renamed to `camel-archetype-webconsole`.

camel-spring-tests and camel-blueprint-tests artifacts

Since Apache Camel 2.8, the `camel-spring-tests` and the `camel-blueprint-tests` Maven artifacts are no longer being released to the Maven central repository.

pollEnrich

Since Apache Camel 2.8, `pollEnrich` now sets an empty message body, if the Content Enricher[8] could not poll from the resource. Previously the old message body would be preserved.

Since Apache Camel 2.10, the `pollEnrich` DSL command now blocks, if no messages are available and no timeout has been configured.

Custom TypeConverter

Since Apache Camel 2.8, it is highly recommended that you modify the contents of the `META-INF/services/org/apache/camel/TypeConverter` file to specify the *fully-qualified names* of the type converter classes. This is much more efficient, because it makes it unnecessary to scan all of the Java packages to find the type converter classes at run time.

Camel test component

Since Apache Camel 2.9, debugger is now disabled by default in Camel Test. You would need to override the `isUseDebugger()` method and return `true` to enable it.

Simple language

Since Apache Camel 2.9, the Simple language has an improved syntax parser. Simple is now more strict and will report syntax errors for invalid input. For example, predicates now require literal text to be enclosed in quotes. The `range` operator also requires the range to be enclosed in quotes. For details, see "The Simple Language" in *Routing Expression and Predicate Languages*.

Since Apache Camel 2.10, the simple language no longer trims white space surrounding an expression in the Java DSL. On the other hand, the simple language does trims white space surrounding an expression in the XML DSL

[8] http://camel.apache.org/content-enricher.html

by default, but you can disable this behavior by setting `trim="false"` in the `simple` element.

URI normalization

Since Apache Camel 2.10, URI normalization now recognizes pre-existing `%nn` decimal encodings in URI strings.

Intercept

When using Apache Camel interceptors, the behavior when combining the `skipSendToOriginalEndpoint()` clause with the `when()` predicate has changed in Apache Camel 2.10.

Now, when you combine the `skipSendToOriginalEndpoint()` clause with a `when()` predicate, the original endpoint is skipped, only if the `when()` predicate evaluates to `true`. For example, in the following route, the `mock:result` endpoint will be skipped, only if the message body is equal to `Hello World`:

```
interceptSendToEndpoint("mock:foo")
    .skipSendToOriginalEndpoint()
    .when(body().isEqualTo("Hello World"))
    .to("mock:detour").transform(constant("Bye World"));

from("direct:second")
    .to("mock:bar")
    .to("mock:foo")
    .to("mock:result");
```

In previous versions, the original endpoint would have been skipped, even if the `when()` predicate evaluated to `false`.

Thread name pattern

Since Apache Camel 2.10, the default pattern for thread names is now:

```
Camel (#camelId#) thread ##counter# - #name#
```

In previous releases the thread name pattern was:

```
Camel (${camelId}) thread #${counter} - ${name}
```

Type converters

Since Apache Camel 2.10, the setting `lazyLoadTypeConverter=true` is now deprecated and will be removed in a future release. The recommended practice is to load type converters at start up time.

The Apache Camel test kit no longer lazily loads type converters.

Apache Camel now fails faster during type conversion, by throwing `TypeConversionException` to the caller.

Apache Camel no longer supports using `java.beans.PropertyEditor` for type conversion. This approach is slow, not thread safe, and uses third party JARs on the classpath, which can cause unwanted side effects.

MDC logging

Since Apache Camel 2.10, the keys for MDC logging are now prefixed with `.camel`.

Chapter 4. Apache CXF Issues

Fuse ESB Enterprise 7.1.0.fuse-047 uses Apache CXF 2.6. Since the last release, Apache CXF has been upgraded from version 2.5 to version 2.6. This introduces a few migration issues. In particular, you should note that the `cxf-bundle` artifact (which encapsulated the complete Apache CXF runtime) is not supported in this release.

Security Caching Changes

EhCache used by default	`EhCache` is now used by default to cache `SecurityToken` tokens for re-use, and for replay detection. It is possible to plug in other implementations by configuration.
Replay detection	The WS-Security module now supports replay detection of timestamps and `UsernameToken` nonces by default . The default caching time is 60 minutes.
Issued tokens default lifetime	The STS now issues SAML and `SecurityContextToken` tokens with a default lifetime of 30 minutes (they are also stored in the cache for this length of time).

Runtime Changes

WS-SecurityPolicy syntax enforcement

The syntax of WS-SecurityPolicy policies is enforced more strictly. Some policies that worked with versions of Apache CXF prior to 2.6 will not load in CXF 2.6 as a result.

JMS transport message format

When using the `TextMessage` message format, Apache CXF now leaves the contents as a `String` and uses `java.io.Reader` and `java.io.Writer` to boost performance. Previously, Apache CXF would convert a `String` to or from `byte[]`, which required the use of encoders, increasing memory usage, and so on. Some interceptors and features of Apache CXF may expect or require the `InputStream` and `OutputStream` types, instead of the `Reader` and `Writer` types. In this case, you may need to use the `BytesMessage` message format instead.

API Changes

Modules removed

The following modules have been removed from Apache CXF 2.6:

`cxf-common-utilities`
: Merged into the `cxf-api` module.

`cxf-rt-binding-http`
: This module has been deprecated for some time and its functionality has long been replaceable with the JAX-RS front-end.

Classes removed

The following classes have been removed from Apache CXF 2.6:

`org.apache.cxf.jaxrs.ext.codegen.CodeGeneratorProvider`
: Please use a `wadl2java` code-generator to generate the JAX-RS code.

`org.apache.cxf.jaxrs.features.clustering.FailoverFeature`
: Please use the common `org.apache.cxf.clustering.FailoverFeature` class instead.

Classes moved

To resolve some of the split-package issues between JAR file, a few classes have been moved to a different Java package:

- `JAXButils` has moved from the `org.apache.cxf.jaxb` package to the `org.apache.cxf.common.jaxb` package.
- Many of the internal `Impl` classes and `Manager` classes (for example, `BindingFactoryManagerImpl`, `CXFBusLifeCycleManager`, and so on) have moved into the `org.apache.cxf.bus.managers` package. In any case, you should not reference these implementation classes directly, but instead use the interfaces that they implement.

Classes changed

The following classes have been modified in Apache CXF 2.6:

All API methods that take or return *generic* classes

: All API methods that take or return Java generic classes have been updated to define the generic part properly. For example, methods like `Class getServiceClass()` have been updated to `Class<?> getServiceClass()`.

`AbstractConduitSelector`

: The `selectedConduit` field of the `AbstractConduitSelector` class has been removed, because a `ConduitSelector` might be used to select *multiple* conduits in certain scenarios and the `selectedConduit` field is not always guaranteed to reflect the currently selected conduit.

`org.apache.cxf.tools.common.DataTypeAdapter`

: The `DataTypeAdapter` class has been deprecated and moved to `org.apache.cxf.xjc.runtime.DataTypeAdapter` in a new runtime JAR that belongs to the `cxf-xjc` package. This enables you to use the runtime without pulling in all of the Apache CXF tooling dependencies. The `DataTypeAdapter` does not have any other Apache CXF dependencies and thus can be used outside Apache CXF as well.

XJC ToString plugin

The XJC `ToString` plugin has had its runtime dependencies moved out of the `org.apache.cxf.tools` package and into the `cxf-xjc-runtime` JAR file. If you use the newer version of the `ToString` plugin, you will need to add the `cxf-xjc-runtime` dependency to your application. However, the `cxf-xjc-runtime` JAR file does not depend on other Apache CXF JAR files or classes and can thus easily be used in other applications, without pulling in as many dependencies.

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>2.6.0.fuse-71-047</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>2.6.0.fuse-71-047</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
          <artifactId>cxf-rt-transports-http-jetty</artifact
Id>
            <version>2.6.0.fuse-71-047</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>
```

org.apache.cxf.tools.* classes moved

The `org.apache.cxf.tools.*` classes that were in the `cxf-api` JAR have been moved into the `cxf-tools-common` JAR or the `cxf-tools-validator` JAR.

org.apache.cxf.ws.policy classes moved

The `org.apache.cxf.ws.policy` classes that were in the `cxf-api` JAR have been moved into the `cxf-rt-ws-policy` JAR.

cxf-common-utilities classes moved

The `cxf-common-utilities` JAR is no longer available. All the classes from that JAR have been moved into the `cxf-api` JAR.

cxf-rt-core and cxf-rt-ws-addr classes moved

Various classes in the `cxf-rt-core` JAR and the `cxf-rt-ws-addr` JAR have been moved into the `cxf-api` JAR in order to resolve split-package issues. Formerly, dependencies on `cxf-rt-core` would have transitively pulled in the `cxf-api` JAR anyway, so there should be little impact.

Spring now optional

Spring is now an optional component of the `http-jetty` transports module and other modules. If you have any applications that rely on pulling in Spring transitively through Apache CXF (that is, no explicit dependency on Spring is declared), you will now have to modify the Maven POM files to add an explicit dependency on Spring.

JAX-RS provider moved

Most of the optional JAX-RS providers have been moved out of the `cxf-rt-frontend-jaxrs` module and into the `cxf-rt-rs-extension-providers` module, with the various dependencies marked as either `optional` or `provided`. Applications that use optional providers now need to add the required dependencies *explicitly* to their POM files. Also, the package names of those providers have changed in order to resolve split-package issues.

For example:

- `org.apache.cxf.jaxrs.provider.JSONProvider` has moved to `org.apache.cxf.jaxrs.provider.json.JSONProvider`.
- `org.apache.cxf.jaxrs.provider.aegis` is the new package for Aegis providers.
- `org.apache.cxf.jaxrs.provider.xmlbeans` is the new package for XMLBeans providers.
- `org.apache.cxf.jaxrs.provider.atom` is the new package for Atom providers.

EhCache now a dependency of cxf-rt-ws-security

EhCache is now a compile-time dependency of the `cxf-rt-ws-security` module in order to support caching and replay detection. It can be safely excluded downstream, at the expense of weakening the caching support.

CORS package renamed and moved

The CORS package has changed to `org.apache.cxf.rs.security.cors` and moved to the new `cxf-rt-rs-security-cors` module.

JAX-RS search extension moved

JAX-RS search extension code has been moved to the new `cxf-rt-rs-extension-search` module.

Chapter 5. Migrating from Fuse HQ to JBoss Operations Network

While Fuse HQ and JBoss Operations Network share the same heritage, they have some important differences.

Overview

The major differences between Fuse HQ and JBoss Operations Network(JBoss ON) include:

- The resource tree is flatter
- JBoss ON server and agent installations are separate
- Database support
- JBoss ON has better automatic discovery
- Different Java APIs
- Different scripting languages

Database support

Regardless of the database you are currently using, JBoss ON will require you to create all new tables for the monitoring data as part of the server installation process.

JBoss ON supports the following databases:

Database	Version
Oracle	11g R2 RA 11g R2 11g R1 RAC 11g R1 10g R2 *(deprecated)* 10g R1 *(deprecated)*
PostgreSQL	9.1.x

Database	Version
	9.0.x 8.4.x 8.3.x

Installing	To install JBoss ON, download the server from the Red Hat Customer Portal[1] and follow the instructions for installing on your platform. See the Installation Guide[2]. JBoss ON agents and monitoring plug-ins are installed from the JBoss ON Web UI.
Configuration	JBoss ON agents and servers are configured differently from Fuse HQ agents and servers. For details see the Configuring JBoss ON Servers and Agents[3].
Scripting	JON scripting is different from Fuse HQ scripting. To update your scripts see the Writing JBoss ON Command-Line Scripts[4].
More information	For more information see the JBoss ON Documentation[5].

[1] https://access.redhat.com/downloads/
[2] https://access.redhat.com/knowledge/docs/en-US/JBoss_Operations_Network/3.1/html/Installation_Guide/index.html
[3] https://access.redhat.com/knowledge/docs/en-US/JBoss_Operations_Network/3.1/html/Admin_Configuring_JON_Servers_and_Agents/index.html
[4] https://access.redhat.com/knowledge/docs/en-US/JBoss_Operations_Network/3.1/html/Dev_Writing_JON_Command-Line_Scripts/index.html
[5] https://access.redhat.com/knowledge/docs/JBoss_Operations_Network/