

FUSE™ ESB

Migrating to FUSE™ ESB 4.0

Version 4.0
October 2008

Migrating to FUSE™ ESB 4.0

Version 4.0

Published 26 Jan 2009

Copyright © 2008 IONA Technologies PLC, a wholly-owned subsidiary of Progress Software Corporation.

Legal Notices

Progress Software Corporation and/or its subsidiaries may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this publication. Except as expressly provided in any written license agreement from Progress Software Corporation, the furnishing of this publication does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Any rights not expressly granted herein are reserved.

Progress, IONA, IONA Technologies, the IONA logo, Orbix, High Performance Integration, Artix, FUSE, and Making Software Work Together are trademarks or registered trademarks of Progress Software Corporation and/or its subsidiaries in the US and other countries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the US and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate Progress Software Corporation makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Progress Software Corporation shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third-party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this publication. This publication and features described herein are subject to change without notice. Portions of this document may include Apache Foundation documentation, all rights reserved.

Table of Contents

| | |
|---|----|
| I. Differences Between Version 3.x and Version 4.0 | 9 |
| Feature Disparities | 11 |
| Configuration | 13 |
| API Changes | 15 |
| JBI Issues | 17 |
| FUSE Services Framework Issues | 19 |
| II. Migrating Functionality to Version 4.0 | 21 |
| Migrating JAX-WS Services to Native FUSE ESB Applications | 23 |
| JMS Flows and Clustering | 27 |

DRAFT

DRAFT

List of Figures

| | |
|---|----|
| 1. Using JMS Endpoints to Make a JMS Flow | 27 |
|---|----|

DRAFT

DRAFT

List of Examples

| | |
|---|----|
| 1. Importing MessageExchangeListener | 15 |
| 2. Using the Maven java2ws Plug-In | 19 |
| 3. FUSE Services Framework Code Generation Plug-ins | 24 |
| 4. OSGi Bundle Plug-in Configuration | 25 |
| 5. OSGi Bundle Plug-in Configuration | 25 |
| 6. FUSE Services Framework Spring Configuration | 26 |

DRAFT

DRAFT

Part I. Differences Between Version 3.x and Version 4.0

Version 4.0 has a number of functional differences from previous versions.

| | |
|---|-----------|
| Feature Disparities | 11 |
| Configuration | 13 |
| API Changes | 15 |
| JBI Issues | 17 |
| FUSE Services Framework Issues | 19 |

DRAFT

Draft

DRAFT

Feature Disparities

FUSE ESB 4.0 is a complete re-implementation. It shares very little code with earlier versions. While FUSE ESB 4.0 retains most of the functionality of earlier versions, there are a number of issues you will encounter when moving to version 4.0.

Clustering

There is no longer a central way to configure a cluster of FUSE ESB containers. You can create a cluster using explicit JMS endpoints that shuttle messages between the members of a cluster. For more information see [JMS Flows and Clustering on page 27](#).

Flows

FUSE ESB 4.0 no longer uses *flows* for message exchanges. Applications that relied on specific features of the JMS flow or the JCA flow will need to find other ways to access those features.

Hot deployment

The default name of the hot deployment folder has been changed to `deploy`. You can change the location of the hot deployment folder by editing the `org.apache.servicemix.filemonitor.monitorDir` in `etc/config.properties`.

The default scan interval is 500ms. You can change the interval by editing the `org.apache.servicemix.filemonitor.scanInterval` in `etc/config.properties`.

Transactions

Transactions now work the same way for both synchronous and asynchronous JBI exchanges. The transaction is conveyed with the exchange in all cases. The resulting behavior is the same as using synchronous sends in previous versions of FUSE ESB.

DRAFT

Configuration

Most of the configuration has been completely altered for FUSE ESB 4.0. This makes the job of configuring the FUSE ESB runtime easier.

Container XML configuration

The container no longer uses Spring XML configuration. Configuration is done using per-bundle properties files.

JMX configuration

The JMX configuration has been moved from the `service.properties` file to the `org.apache.servicemix.management.cfg` file.

For more information see [Changing the JMX Management Properties](#) in the *Managing the FUSE™ ESB Runtime*.

Logging configuration

The logging configuration has been moved from the `log4j.xml` file to the `org.ops4j.pax.logging.cfg` file.

For more information see [Logging Configuration](#) in the *Managing the FUSE™ ESB Runtime*

DRAFT

API Changes

Some classes in FUSE ESB version 3.x have been moved in FUSE ESB version 4.0.

Overview

A number of classes been moved during the upgrade to version 4.0. In most cases the classes have simply been packaged into different jars. In some cases, the classes have been moved to new packages.

New jars

The `servicemix-core` jar and the `servicemix-jbi` jar have been removed from FUSE ESB 4.0. The classes that were packaged in those jars have been moved to the `servicemix-common` jar and the `servicemix-utils` jar.

If your projects have explicit dependencies on either of these artifacts, you need to update it to depend on the new artifacts.

Message exchange listeners

If your application uses a custom implementation of the `MessageExchangeListener` interface you will need to update your code. The interface has been moved from the `org.apache.servicemix` package to the `org.apache.servicemix.jbi.listener` package. [Example 1 on page 15](#) shows the import statement for the `MessageExchangeListener` interface.

Example 1. Importing MessageExchangeListener

```
import org.apache.servicemix.jbi.listener.MessageExchangeL
istener
```

DRAFT

JB1 Issues

FUSE ESB 4.0 maintains full JBI compliance. There are a few differences between 4.0 and previous versions.

OSGi Components

All of the JBI components included in FUSE ESB 4.0 are packaged as OSGi bundles. This means that:

- Applications do not need to use the JBI packaging to use the JBI components.



Note

Service units packaged as OSGi bundles will be subject to JBI lifecycle semantics.

- Service units can be deployed using a simple XML file.
 - Endpoints deployed in JBI service units can communicate with OSGi services.
-

Component Configuration

All of JBI component configuration that used to reside in `installDir/conf/components.properties` are now configured using per-component configuration files. For example, the component configuration for the JMS component is located in `installDir/etc/servicemix-jms.cfg`.

In addition, each component's thread pool can be configured independent of the other components. Each component's configuration file includes a `threadPoolCorePoolSize` property and a `threadPoolMaximumPoolSize` property. The `threadPoolCorePoolSize` property specifies the minimum number of threads in a component's thread pool at start-up. The `threadPoolMaximumPoolSize` property specifies the maximum number of threads allowed in a component's thread pool.

Deprecated components

The following JBI components are not included in FUSE ESB 4.0:

- lightweight container

- jsr181
-

Tooling

All of the Maven tooling from previous versions of FUSE ESB are included with 4.0.

The only limitations are the following:

- the Maven plug-in can no longer deploy JBI artifacts to the container
 - the Maven plug-in can no longer ensure that the required JBI artifacts for a project are running in the container
-

Packaging

Older versions of the JBI tooling generated JAR files for service assemblies and only converted them to ZIP files when they were deployed. Version 4.0 of the tooling creates ZIP files for service assemblies and places them in the assembly project's `target` folder.

Shared libraries

Previous versions of FUSE ESB allowed you to reference shared libraries from within service units. This functionality is not included in 4.0. Service units will have to explicitly include all of the required classes and support files they require.

FUSE Services Framework Issues

FUSE ESB 4.0 uses FUSE Services Framework 2.1. This introduces a few migration issues.

JAXB 2.1 annotations

The code generated by the 4.0 code generators adds some JAXB 2.1 specific annotations. These annotations are not compatible with previous versions of FUSE Services Framework.

WS-Addressing

JAX-WS 2.1 supports WS-Addressing directly in the APIs. WSDLs that use the `EndpointReferenceType` will now generate the JAX-WS 2.1 `EndpointReference` instead of the FUSE Services Framework proprietary type that was generated in FUSE Services Framework 2.0.

You can use the `wsdl2java` command's `-noAddressBinding` option to force the generation of the old-style endpoint reference code.

Tooling

The `java2wsdl` tool and the Maven code generator plug-in's `java2wsdl` goal have been replaced with more generic tools.

The `java2wsdl` tool has been replaced with the `java2ws` tool.

The Maven code generator plug-in's `java2wsdl` goal has been replaced with the new `java2ws` plug-in. [Example 2 on page 19](#) shows an example of using the new plug-in.

Example 2. Using the Maven java2ws Plug-In

```
<project>
...
<plugin>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-java2ws-plugin</artifactId>
  <version>2.1</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-frontend-jaxws</artifactId>
      <version>2.1</version>
    </dependency>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-rt-frontend-simple</artifactId>
      <version>2.1</version>
    </dependency>
  </dependencies>
</plugin>
</project>
```

```
</dependency>
</dependencies>
<executions>
  <execution>
    <id>generate-test-sources</id>
    <phase>generate-test-sources</phase>
    <configuration>
      <className>org.apache.hello_world.Greeter</className>

      <genWsd1>true</genWsd1>
      <verbose>true</verbose>
    </configuration>
    <goals>
      <goal>java2ws</goal>
    </goals>
  </execution>
</executions>
</plugin>
...
</project>
```

ASM

The JAX-WS frontend now requires ASM 2.x or 3.x to be able to process some of the JAXB annotations on the SEI. There can be conflicts with other applications, such as Hibernate, that use asm.

If you don't use those annotations on the SEI or if you have generated wrapper classes, you can remove `asm jar` from your installation.

If you use the annotations, the workaround for Hibernate is to remove Hibernate's ASM 1.x jar used by it and replace Hibernate's `cglib jar` with the `cglib-nodeps jar` that includes a special internal version of ASM that would not conflict with the 2.x/3.x version used by FUSE Services Framework.

Part II. Migrating Functionality to Version 4.0

Most of your applications will migrate effortlessly from Version 3.x to 4.0. There are instances, however, where you will need to do some work.

| | |
|--|-----------|
| Migrating JAX-WS Services to Native FUSE ESB Applications | 23 |
| JMS Flows and Clustering | 27 |

DRAFT

Draft

DRAFT

Migrating JAX-WS Services to Native FUSE ESB Applications

FUSE ESB 4 makes it possible to deploy JAX-WS services as native applications without wrapping them as JBI endpoints. Services developed this way have direct access to the containers transport layer.

Overview

You can continue to deploy JAX-WS services using the FUSE Services Framework JBI components, however using the native integration of FUSE Services Framework with FUSE ESB makes JAX-WS services easier to maintain. The JBI components require that you develop at least two separate endpoints for each service. Using the native integration, JAX-WS services are developed as a single application. Native JAX-WS services also do not need to interact with the NMR which makes them easier to debug.

Migrating a JBI based JAX-WS service to a native application is straight-forward. The code and WSDL from the JBI implementation do not need to change. You will, however, need to modify the following:

- [the project's file structure](#)
 - [the project's POM](#)
 - [the service's Spring configuration](#)
-

The project structure

JBI projects for a JAX-WS service contain at least three subprojects:

- the service engine service unit
- the binding component service unit
- the service assembly

JAX-WS services using the native FUSE Services Framework integration are built as a single project. Your project folder only needs a top-level POM and a `src` folder.

The source code from the JBI project's service engine project should be moved under the `src` folder. In addition, the WSDL from the JBI project should be moved to `src/main/resources/`.

The POM

You cannot migrate your POM file from version 3.x to version 4.0. You will need to do the following:

1. Create a new top-level POM that includes all of the dependencies for a FUSE ESB project.
2. Set the `packaging` element to `bundle`.
3. If your JBI project's service engine used the FUSE Services Framework code generation plug-ins, you will need to move them to your new top-level POM.

The entry for the FUSE Services Framework code generation plug-ins will be similar to the entry shown in [Example 3 on page 24](#).

Example 3. FUSE Services Framework Code Generation Plug-ins

```
<plugin>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-codegen-plugin</artifactId>
  <version>${cxf.version}</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <configuration>
        ...
      </configuration>
      <goals>
        ...
      </goals>
    </execution>
  </executions>
</plugin>
```

4. Move the compilation and JAX-WS dependencies to your new top-level POM.
5. Add the OSGi bundle plug-in to you new top-level POM.

[Example 4 on page 25](#) shows the entry for the OSGi bundle plug-in.

Example 4. OSGi Bundle Plug-in Configuration

```

<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
    <instructions>
      <Bundle-SymbolicName>bundleId</Bundle-SymbolicName>
      <Import-Package>importList</Import-Package>
      <Private-Package>currentPackage</Private-Package>
      <Require-Bundle>org.apache.cxf.cxf-bundle</Require-Bundle>
    </instructions>
  </configuration>
</plugin>

```

- *bundleId* is the identifier used by the container for the bundle.
 - *importList* is a comma-separated list of packages the application needs to import from other bundles.
6. Add the required bundle dependencies to the OSGi bundle plug-in.

[Example 5 on page 25](#) shows the minimum list of packages that a native FUSE Services Framework application needs.

Example 5. OSGi Bundle Plug-in Configuration

```

javax.jws,
javax.wsdl,
META-INF.cxf,
META-INF.cxf.osgi,
org.apache.cxf.bus,
org.apache.cxf.bus.spring,
org.apache.cxf.bus.resource,
org.apache.cxf.configuration.spring,
org.apache.cxf.resource,
org.springframework.beans.factory.config

```

The configuration

FUSE Services Framework JBI projects have their configuration spread over two service units' `xbean.xml` files. The service implementation's configuration is specified in the service engine's service unit using an `cxfse:endpoint` element. The transport's configuration is specified in the binding component's service unit using an `cxfbc:consumer` element.

Native FUSE Services Framework projects have their configuration located in a single Spring configuration file. This file is located in the `src/main/resources/META-INF/spring`. The FUSE Services Framework service is configured using the standard FUSE Services Framework configuration elements.

[Example 6 on page 26](#) shows a sample configuration for a FUSE Services Framework service.

Example 6. FUSE Services Framework Spring Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframe
work.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

  <import resource="classpath:META-INF/cxf/cxf.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-http.xml" />
  <import resource="classpath:META-INF/cxf/osgi/cxf-extension-osgi.xml" />

  <jaxws:endpoint id="PersonService"
    implementor="org.apache.servicemix.samples.wsdl_first.PersonImpl"
    address="/PersonService"
    wsdlLocation="classpath:person.wsdl"/>
</beans>
```

More Information

For more information on using the native JAX-WS integration see [Developing and Deploying JAX-WS Services with FUSE™ ESB](#).

JMS Flows and Clustering

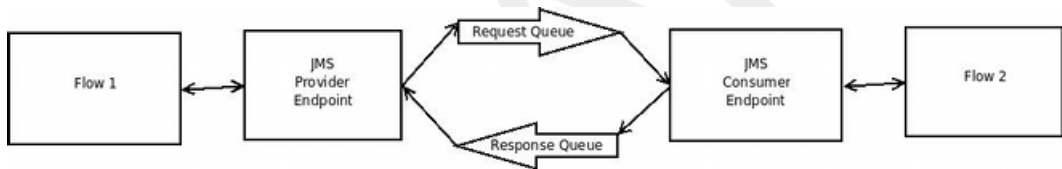
In FUSE ESB 3 clustering was implemented using the JMS flow. You can accomplish similar results using hard wired JMS endpoints as part of your flow. The JMS destinations used by the endpoints can be used to provide cross-container communication.

Overview

The JMS flow in FUSE ESB 3 allowed users to gain a level of persistence and failure tolerance out of their applications. The JMS flow used the internal FUSE Message Broker to manage temporary JMS destinations used for message exchanges. The JMS flow also made it possible to create a cluster of FUSE ESB containers over which you could distribute the endpoints used in an application.

While the JMS flow is not present in FUSE ESB 4, you can get some of the same benefits by using hard coded JMS endpoints along your flow as shown in [Figure 1 on page 27](#).

Figure 1. Using JMS Endpoints to Make a JMS Flow



Where ever you place the JMS endpoints, the message exchange will be passed into a JMS destination that you can set up with any persistence level desired. You can also use the JMS bridge to distribute your endpoints among a number of different containers.

Limitations

While this work around provides a number of the benefits of using JMS flows and clustering it does have the following limitations:

- You must use named JMS destinations.
- Messages are only written out to JMS destinations between the JMS endpoints.
- You cannot fully distribute a message flow among a number of containers. A flow can only span multiple containers when traveling from one JMS endpoint to another.

- Containers in a cluster established using dedicated JMS endpoints are not a true cluster. The containers have no knowledge of the endpoints in the remote containers.

More information

For more information on using JMS endpoints see [Using the JMS Binding Component](#).

DRAFT