CONTENTS INCLUDE:

Updated for ServiceMix 4.2

ServiceMix 4.2

The Apache Open Source ESB

By Jos Dirksen

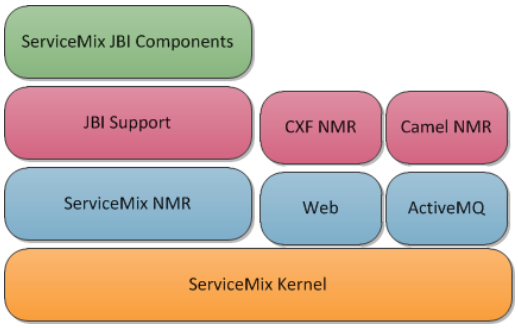
ABOUT SERVICEMIX 4.2

In the open-source community, there are many different solutions for each problem. When you look for an open-source ESB, however, you don't have that many options. Even though there are many open-source ESB projects, not all of them are mature enough to be used to solve enterprise mission-critical integration problems. ServiceMix is one of the open-source projects that is mature enough to be used in these scenarios. ServiceMix, an Apache project, has been around for more than five years now. It provides all the features you expect from an ESB such as routing, transformation, etc. The previous version was built based on JBI (JSR-208), but in its latest iteration, which we're discussing in this Refcard, ServiceMix has moved to an OSGi-based architecture, which we'll discuss later on.

This DZone Refcard will provide an overview of the core elements of ServiceMix 4.2 and will show you how to use ServiceMix 4.2 by providing example configurations.

SERVICEMIX 4.2 ARCHITECTURE

Before we show how to configure ServiceMix 4.2 for use, let us first look at the architecture of ServiceMix 4.2.



This figure shows the following components:

ServiceMix Kernel: In this figure, you can see that the basis of ServiceMix 4.2 is the ServiceMix Kernel. This kernel, which is based on the Apache Karaf project (an OSGi based runtime), handles the core features ServiceMix provides, such as hot-deployment, provisioning of libraries or applications, remote access using ssh, JMX management, and more.

ServiceMix NMR: This component, a normalized message router, handles all the routing of messages within ServiceMix and is used by all the other components.

ActiveMQ: ActiveMQ, another Apache project, is the message broker that is used to exchange messages between components. Besides this, ActiveMQ can also be used to create a fully distributed ESB.

Web: ServiceMix 4.2 also provides a Web component. You can use this to start ServiceMix 4.2 embedded in a Web application. An example of this is provided in the ServiceMix distribution.

JBI Support: The previous version of ServiceMix was based on JBI 1.0. For JBI, a lot of components (from ServiceMix, but also from other parties) are available. ServiceMix 4.2 has full support for JBI, this way you can still use all the components provided by the 3.x version of ServiceMix. For the best results, you should use the 2010.01 version of these components.

Camel NMR: ServiceMix 4.2 provides a couple of different ways you can configure routing. You can use the endpoints provided by the ServiceMix NMR, but you can also use more advanced routing engines. One of those is the Camel NMR. This component allows you to run Camel based routes on ServiceMix.

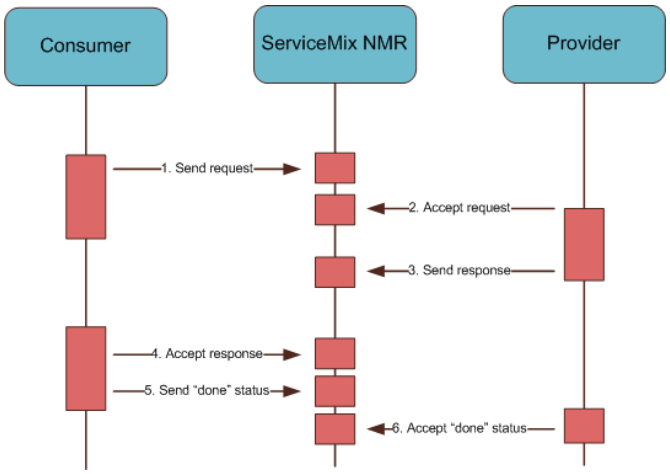
CXF NMR: Besides an NMR based on Camel, ServiceMix also provides an NMR based on CXF. You can use this NMR to expose and route to Java POJOs annotated with JAX-WS annotations.

Hot Tip

OSGi Runtime

ServiceMix runs on an OSGi-based kernel, but what is OSGi? In short, an OSGi container provides a service based in-VM platform on which you can deploy services and components dynamically. OSGi provides strict class loading separation and forces you to think about the dependencies your components have. Besides that, OSGi also defines a simple lifecycle model for your services and components. This results in an environment where you can easily add and remove components and services at runtime and allows the creation of modular applications. An added advantage of using an OSGi container is that you can use many components out of the box: remote administration, a Web container, configuration and preferences services, etc.

Before we move on to the next part, let's have a quick look at how a message is processed by ServiceMix. The following figure shows how a message is routed by the NMR. In this case, we're showing a reply/response (in-out) message pattern.



In this figure, you can see a number of steps being executed:

1. The consumer creates a message exchange for a specific service and sends a request.
2. The NMR determines the provider this exchange needs to be sent to and queues the message for delivery. The provider accepts this message and executes its business logic.
3. After the provider has finished processing, the response message is returned to the NMR.
4. The NMR once again queues the message for delivery, this time to the consumer. The consumer accepts the message.
5. After the response is accepted, the consumer sends a confirmation to the NMR.
6. The NMR routes this confirmation to the provider, who accepts it and ends this message exchange.

Now that we've seen the architecture and how a message is handled by the NMR, we'll have a look at how to configure ServiceMix 4.2.

CONFIGURATION OF SERVICEMIX 4.2

ServiceMix 4.2 configuration is mostly done through Spring XML files supported by XML schemas for easy code completion. Let's look at two simple examples. The first one uses the File Binding component to poll a directory and the second one exposes a Web service using ServiceMix's CXF support.

```
<beans xmlns:file="http://servicemix.Apache.org/file/1.0"
              xmlns:dzone="http://servicemix.org/dzone/">
  <file:poller service="foo:filePoller"
              endpoint="filePoller"
              targetService="foo:fileSender"
              file="inbox" />
</beans>
```

In this listing, you can see that we define a `poller`. A poller is one of the standard components that is provided by ServiceMix's file-binding-component. If we deploy this configuration to ServiceMix, ServiceMix will start polling the inbox directory for files. If it finds one, the file will be sent to the specified `targetService`.

Hot Tip

Service Addressing

An important concept to understand when working with ServiceMix is that of services and endpoints. When you configure services on a component, you need to tell ServiceMix how to route messages to and from that service. This name is called a service endpoint. If you look back at the previous example, we created a file:poller. On this file:poller we defined a service and an endpoint attribute. These two attributes together uniquely identify this file:poller. Note though that you can have multiple endpoints defined on the same service. You can also see a targetService attribute on the file:poller. Besides this attribute, there is also a targetEndpoint attribute. With these two attributes, you identify the service endpoint to sent the message to. The targetEndpoint isn't always needed, if only one endpoint is registered on that service.

In the following listing, we've again used a simple XML file. This time we've configured a Web service.

```
<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.springframework.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" /> 1
    <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="helloWorld"
                      implementor="dzone.refcards.HelloWorld"
                      address="/HelloWorld"/>
</beans>
```

In this listing, we use a `jaxws:endpoint` to define a Web service. The implementor points to a simple POJO annotated with JAX-WS annotations. If this example is deployed to ServiceMix, ServiceMix will register a Web service based on the value in the `address` attribute.

This is just one way to expose Web services using ServiceMix 4.2. You could also use the servicemix-cxf-bc component for this.

DEPLOYMENT OF SERVICEMIX 4.2 COMPONENTS

ServiceMix provides a number of different options that you can use to deploy artifacts. In this section, we'll look at these options and show you how to use these.

ServiceMix 4.2, Deployment Options

Name	Description
OSGi Bundles	ServiceMix 4.2 is built around OSGi and ServiceMix 4.2 also allows you to deploy your configurations as an OSGi bundle with all the advantages OSGi provides.
Spring XML files	ServiceMix 4.2 support plain Spring XML files.
JBI artifacts	You can also deploy artifacts following the JBI standard (service assemblies and service units) to ServiceMix 4.2.
Feature descriptors	This is a Karaf specific way for installing applications. It will install the necessary OSGi bundles and will add configuration defaults. This is mostly used to install core parts of the ServiceMix distribution.

OSGi Bundle Deployment

The easiest way to create an OSGi-based ServiceMix bundle is by using Maven 2 or 3. To create a bundle, you need to take a couple of simple steps. The first one is adding the maven-bundle-plugin to your `pom.xml` file. This is shown in the following code fragment.

```
...
<dependencies>
  <dependency>
    <groupId>org.Apache.felix</groupId>
    <artifactId>org.osgi.core</name>
    <version>1.0.0</version>
  </dependency>
  ...
</dependencies>
...
<build>
  <plugins>
    <plugin>
      <groupId>org.Apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <configuration>
        <instructions>
          <Bundle-SymbolicName>${pom.artifactId}</Bundle
           SymbolicName>
          <Import-Package>*,org.Apache.camel.osgi</Import-Package>
          <Private-Package>org.Apache.servicemix.examples.camel</
           Private-Package>
        </instructions>
      </configuration>
    </plugin>
  </plugins>
</build>
...
```

The important part here is the `instructions` section. This determines how the plugin packages your project. For more information on these settings, see the maven OSGi bundle plugin page at http://cwiki.Apache.org/FELIX/Apache-felixmaven-bundle-plugin-bnd.html.

The next step is to make sure your project is bundled as a OSGi bundle. You do this by setting the `<packaging>` element in your `pom.xml` to `bundle`.

Now you can use `mvn install` to create an OSGi bundle, which you can copy to the `deploy` directory of ServiceMix and your bundle will be installed. If you use Spring to configure your application, make sure the Spring configuration files are located in the `META-INF/spring` directory. That way, the Spring application context will be automatically created based on these files.

If you don't want to do this by hand, you can also use a Maven archetype. ServiceMix provides a set of archetypes you can use. A good starting point for a project is the Camel OSGi archetype that you can use by executing the following Maven command:

```
mvn archetype:create -DarchetypeGroupId=org.Apache.servicemix.tooling
-DarchetypeArtifactId=servicemix-osgi-camel-archetype
-DarchetypeVersion=2010.01.0-fuse-01-00
-DgroupId=com.yourcompany -DartifactId=camel-router
-DremoteRepositories=http://repo.fusesource.com/maven2/
```

There are many other archetypes available. For an overview of the available archetypes, see:
http://repo.fusesource.com/maven2/org/apache/servicemix/tooling/

Spring XML Files Deployment

It's also possible to deploy Spring files without OSGi. Just drop a Spring file into the deploy directory. There are two points to take into account. First, you need to add the following to your Spring configuration file:

```
<bean class="org.Apache.servicemix.common.osgi.EndpointExporter" />
```

This will register the endpoints you've configured in your Spring file. The next element is optional but is good practice to add:

```
<manifest>
   Bundle-Version = 1.0.0
   Bundle-Name = Dzone :: Dzone test application
   Bundle-SymbolicName = dzone.refcards.test
   Bundle-Description = An example for servicemix refcard
   Bundle-Vendor = jos.dirksen@gmail.com
   Require-Bundle = servicemix-file, servicemix-eip
</manifest>
```

Using a manifest configuration element allows you to specify how your application is registered in ServiceMix.

JBI Artifacts Deployment

If you've already invested in JBI-based applications, you can still use ServiceMix 4.2 to run them in. Just deploy your Service Assembly (SA) in the ServiceMix deploy directory and ServiceMix will deploy your application.

Feature Descriptor-Based Deployment

If you've got an application that contains many bundles and that requires additional configuration you can use a feature to easily manage this. A feature contains a set of bundles and configuration which can be easily installed from the ServiceMix console. The following listing shows the feature descriptor of the nmr component.

```
<feature name="nmr" version="1.2.0">
  <feature>document</feature>
  <!-- those two bundles are currently required for servicemix-
utils to resolve -->
  <bundle>mvn:org.apache.servicemix.specs/org.apache.servicemix.
specs.activation-api-1.1/1.4.0</bundle>
  <bundle>mvn:org.apache.servicemix.specs/org.apache.servicemix.
specs.jbi-api-1.0/1.4.0</bundle>
  <bundle>mvn:org.apache.servicemix.specs/org.apache.servicemix.
specs.stax-api-1.0/1.4.0</bundle>
  <bundle>mvn:org.apache.servicemix/servicemix-utils/1.2.1</
bundle>
  <bundle>mvn:org.apache.servicemix.document/org.apache.
servicemix.document/1.2.0</bundle>
  <bundle>mvn:org.fusesource.commonman/commons-management/1.0</
bundle>
  <bundle>mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.
api/1.2.0</bundle>
  <bundle>mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.
core/1.2.0</bundle>
  <bundle>mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.
osgi/1.2.0</bundle>
  <bundle>mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.
spring/1.2.0</bundle>
  <bundle>mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.
commands/1.2.0</bundle>
  <bundle>mvn:org.apache.servicemix.nmr/org.apache.servicemix.nmr.
management/1.2.0</bundle>
</feature>
```

If you want to install this feature you can just type `features;install nmr` from the ServiceMix console.

If you don't see a command line when you started ServiceMix use ssh to connect to your local instance. You can do this by using the following command: ssh -l smx -p 8101 localhost and when asked for a password use 'smx'.

ROUTING IN SERVICEMIX 4.2

For routing in ServiceMix, you've got two options:

- **EIP:** ServiceMix provides a JBI component that implements a number of Enterprise Integration Patterns.
- **Camel:** You can use Camel routes in ServiceMix. Camel provides the most flexible and exhaustive routing options for ServiceMix

EIP Component Routing

This routing is provided by the EIP component. To check whether this is installed in your ServiceMix runtime, you can execute `features;list` from the ServiceMix command line. This will show you a list of installed features. If you see `[installed] [2010.01] servicemix-eip` the component is installed. If it shows `uninstalled` instead of `installed`, you can use the `features;install servicemix-eip` to install this component. You can now use this router using a simple XML file:

```
<eip:static-routing-slip service="test:routingSlip"
 endpoint="endpoint">
  <eip:targets>
    <eip:exchange-target service="test:echo" />
    <eip:exchange-target service="test:echo" />
  </eip:targets>
</eip:static-routing-slip>
```

When installed, this component provides the following routing options (this information is also available in the XSD of this component):

XML Element	Description
async-bridge	The async bridge pattern is used to bridge an In-Out exchange with two In-Only (or Robust-In-Only) exchanges. This pattern is the opposite of the pipeline.
content-basedrouter	Component that can be used for content based routing of the message. You can configure this component with a set of predicates which define how the message is routed.
content-enricher	A content enricher can be used to add extra information to the message from a different source.
message-filter	With a message filter you specify a set of predicates which determine whether to process the message or not.
pipeline	The pipeline component is a bridge between an In-Only (or Robust-In-Only) MEP and an In-Out MEP. This is the opposite of the async bridge.
resequencer	A resequencer can be used to re-order a set of incoming messages before passing them on in a the new order.
split-aggregator	A split aggregator is used to reassemble messages that have been split by a splitter.
static-recipient-list	A static recipient list will forward the incoming message to a set of predefined destinations.
static-routing-slip	The static routing slip routes a message through a set of services. It uses the result of the first invocation as input for the next.
wire-tap	The wire-tap will copy and forward a message to the specified destination.
xpath-splitter	This splitter uses an xpath expression to split an incoming message in multiple parts.

Camel Routing

Apache Camel is a project that provides a lot of different routing and integration options. In this section, we'll show how to use Camel with ServiceMix and give an overview of the routing options it provides. Installing the Camel component in ServiceMix is done in the same way as we did for the EIP component. We use the features;list command to check what's already installed and we can use features/add to add new Camel functionality. Once installed, we can use Camel to route messages between our components. Camel provides two types of configuration: XML and Java-based DSL, XML configuration was used for the following two listings:

Camel XML configuration - Listing 1: Camel configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <import resource="classpath:org/Apache/servicemix/camel/nmr/
     camel-nmr.xml" />
      <camelContext xmlns="http://camel.Apache.org/schema/
       spring">
     <route>
       <from uri="ftp://gertv@localhost/testfile?password
        =secret"/>
       <to uri="nmr:IncomingOrders"/>
     </route>
</beans>
```

Camel XML configuration - Listing 2: Target service

```
<beans xmlns:file="http://servicemix.Apache.org/file/1.0"
              xmlns:dzone="http://servicemix.org/dzone/">
   <import resource="classpath:org/Apache/servicemix/camel/nmr/
    camel-nmr.xml" />
   <file:sender service="nmr:IncomingOrders"
            directory="file:target/pollerFiles" />
</beans>
```

In these two listings, you can see how we can easily integrate the Camel routes with the other components from ServiceMix. We use the nmr prefix to tell Camel to send the message to the NMR. The other service, which can be separately deployed, will then pick up this message since it's also configured to listen to a nmr prefixed service.

Now let's look at two listings that use Camel's Java-based DSL to configure the routes. For this, we need a small XML file describing where the routes can be found and a Java file which contains the routing.

Camel Java configuration - Listing 1: Spring configuration

```
<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-2.0.xsd
       http://activemq.Apache.org/camel/schema/spring
http://activemq.Apache.org/camel/schema/spring/camel-spring.xsd ">

<import resource="classpath:org/Apache/servicemix/camel/nmr/camel-
 nmr.xml" />
        <camelContext xmlns="http://activemq.Apache.org/camel/
         schema/spring">
          <package>dzone.refcards.camel.routes</package>
        </camelContext>
</beans>
```

Camel Java configuration - Listing 2: Java route

```
public class SimpleRouter extends RouteBuilder {
    public void configure() throws Exception {

           from("timer:myTimerEvent?fixedRate=true")
                   .setBody(constant("Hello World!")).
                         to("nmr:someService");
    }
}
```

Camel itself provides a lot of standard functionality. It doesn't just provide routing, it can also provide connectivity for different technologies. For more information on Camel, go to http://camel.apache.org/ or see the "Enterprise Integrations Patterns with Camel" Refcard.

SERVICEMIX AND WEB SERVICES

Support for Web services is an important feature for an ESB. ServiceMix uses the CXF project for this. Since CXF is also completely spring based, using CXF to deploy Web services is very easy.

Hosting Web Services

When you want to expose a service as a Web service, you can easily do this using CXF. Just create a CXF OSGi bundle using the archetype: `servicemix-osgicxf-code-first-archetype`. This will create an OSGi- and CXF-enabled maven project that you can use

Differences between ServiceMix and Camel

If you've looked at the Camel website you notice that it provides much the same functionality as ServiceMix. It provides connectivity to various standards and technologies, provides routing and transformation and even allows you to expose Web services. The main difference though is that Camel isn't a container. Camel is designed to be used inside some other container. We've shown that you can use Camel in ServiceMix, but you can also use Camel in other ESBs or in ActiveMQ or CXF. So if you just want an routing and mediation engine Camel is a good choice. If you however need a full ESB with good support for JBI, a flexible OSGi based kernel, hot-deploy and easy administration ServiceMix is the better choice.

to develop Web services. Now just edit the `src/main/resources/META-INF/spring/beans.xml` file and after you've run the mvn install command you can deploy the bundle to ServiceMix. The following listing shows such an example. This will create a Web service and host it on http://localhost:8080/cfx/HelloDzone.

CXF Host Web service example using CXF

```
<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.springframework.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="helloDZone"
             implementor="dzone.examples.ws.HelloDZoneImpl"
                 address="/HelloDzone"/>
</beans>
```

In the previous example, we hosted a Web service that could be called from outside the container. You can also configure CXF to host the Web service internally by prefixing the address with nmr. That way, you can easily expose JAX-WS annotated java beans to the other services inside the ESB. The following example shows this:

CXF Host Web service internally

```
<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.springframework.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/transport/nmr/cxf-
     transportnmr.xml" />
      <jaxws:endpoint id="helloDzone"
             implementor="dzone.examples.ws.HelloDZoneImpl"
                 address="nmr:helloDZone" />
</beans>
```

You can also host a Web services using the `servicemix-cxf-bc` component.

Host Web service using the servicemix-cxf-bc component

```
<beans xmlns:cxfbc="http://servicemix.Apache.org/cxfbc/1.0"
       xmlns:dzone="http://dzone.org/refcard/example">

  <cxfbc:consumer wsdl="classpath:dzone-example.wsdl"
                     targetService="dzone:ExampleService"
                     targetInterface="dzone:Example"/>
</beans>
```

Consuming Web Services

Consuming Web services in ServiceMix is just as easy. ServiceMix provides two different options for this. You can use Camel or use the `servicemix-cxf-bc` component:

Consume Web servicemix using the servicemix-cxf-bc component

```
<beans xmlns:cxfbc="http://servicemix.Apache.org/cxfbc/1.0"
       xmlns:dzone="http://dzone.org/refcard/example">
<cxfbc:provider wsdl="classpath:target-service.wsdl"
                locationURI="http://webservice.com/Service"
                endpoint="ServicePort"
                service="dzone:ServicePortService"/>
</beans>
```

With this configuration, you can consume a Web service that is located at `http://webservice.com/Service` and that is defined by the WSDL file `target-service.wsdl`. Other services can use this component by making a call to the `dzone:ServicePortService`. You can also consume a Web service using Camel. For more information on how you can configure the Camel route, see the Camel CXF integration section of the Camel website: http://camel.apache.org/cxf.html.

For Web services, ServiceMix provides the following useful archetypes:

Name	Description
servicemix-cxf-bc-service-unit	Create a maven project which uses the JBI CXF binding component.
servicemix-cxf-se-service-unit	Create a maven project which uses the JBI CXF service engine.
servicemix-cxf-se-wsdl-firstservice-unit	Create a maven project which uses the JBI CXF service engine. This project is based on WSDL first development.
servicemix-osgi-cxf-code-firstarchetype	Create a maven project which uses CXF and OSGi together. This project is based on code first development.
servicemix-osgi-cxf-wsdl-firstarchetype	Create a maven project which uses CXF and OSGi together. This project is based on wsdl first development.

SERVICEMIX COMPONENTS

Besides integration with Web services through CXF, ServiceMix provides a lot of components you can use out of the box to integrate with various other standards and technologies. In this section, we'll give an overview of these components. This list is based on the 2010.01 versions. Most of this information can also be found in the XML schemas of these components.

ServiceMix Components

XML Element	Description
ServiceMix Bean	
Endpoint	Allows you to define a simple bean that can receive and send message exchanges.
ServiceMix File	
Poller	A polling endpoint that looks for a file or files in a directory and sends the files to a target service. You can configure various options on this endpoint such as archiving, filters, use of subdirectories, etc.
Sender	An endpoint that receives messages from the NMR and writes them to a specific file or directory.
ServiceMix CXF Binding Component	
Consumer	A consumer endpoint that is capable of using SOAP/HTTP or SOAP/JMS.
Provider	A provider endpoint that is capable of exposing SOAP/HTTP or SOAP/JMS services.
ServiceMix CXF Service Engine	
Endpoint	With the Drools Endpoint, you can use a drools rule set as a service or as a router.
ServiceMix FTP	
Poller	This endpoint can be used to poll an FTP directory for files, download them and send them to a service.
Sender	With a sender endpoint, you can store a message on an FTP server.
ServiceMix HTTP	
Consumer	Plain HTTP consumer endpoint. This endpoint can be used to handle plain HTTP request (without SOAP) or to be able to process the request in a non-standard way.
Provider	A plain HTTP provider. This type of endpoint can be used to send non-SOAP requests to HTTP endpoints.
Soap-Consumer	An HTTP consumer endpoint that is optimized to work with SOAP messages.
Soap-Provider	An HTTP provider endpoint that is optimized to work with SOAP messages.
ServiceMix JMS	
Consumer	An endpoint that can receive messages from a JMS broker.
Provider	An endpoint that can send messages to a JMS broker.
Soap-Consumer	A JMS consumer that is optimized to work with SOAP messages.

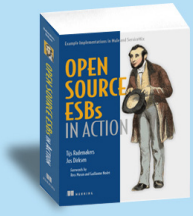
Soap-Provider	A JMS provider that is optimized to work with SOAP messages.
JCA-Consumer	A JMS consumer that uses JCA to connect to the JMS broker.
ServiceMix Mail	
Poller	An endpoint that can be used to retrieve messages.
Sender	An endpoint that you can use to send messages.
ServiceMix OSWorkflow	
Endpoint	This endpoint can be used to start an OSWorkflow process.
ServiceMix Quartz	
Endpoint	The Quartz endpoint can be used to fire messages into the NMR at specific intervals.
ServiceMix Saxon	
XSLT	With the XSLT endpoint, you can apply an XSLT transformation to the received message.
Proxy	The proxy component allows you to transform an incoming message and send it to an endpoint. You can also configure a transformation that needs to be applied to the result of that invocation.
XQuery	The XQuery endpoint can be used to apply a selected XQuery to the input document.
ServiceMix Scripting	
Endpoint	With the scripting endpoint, you can create a service that is implemented using a scripting language. The following languages are supported: Groovy, JRuby, Rhino JavaScript
ServiceMix SMPP	
Consumer	A polling component that binds with jSMPP and receives SMPP messages and sends the SMPPs into the NMR as messages.

Provider	A provider component receives XML message from the NMR and converts into SMPP packet and sends it to SMPP server.
ServiceMix SNMP	
Poller	With this poller, you can receive SNMP events by using the SNMP4J library.
ServiceMix Validation	
Endpoint	With this endpoint, you can provide schema validation of documents using JAXP 1.3 and XMLSchema or RelaxNG.
ServiceMix-VFS	
Poller	A polling endpoint that looks for a file or files in a virtual file system (based on Apache commons-vfs) and sends the files to a target service.
Sender	An endpoint that receives messages from the NMR and writes the message to the virtual file system.
ServiceMix-wsn2005	
Create-pull-point	Lets you create a WS-Notification pull point that can be used by a requester to retrieve accumulated notification messages.
Publisher	Sends messages to a specific topic.
Register-publisher	An endpoint that can be used by publishers to register themselves.
Subscribe	Lets you create subscriptions to a specific topic using the WSNotification specification.
ServiceMix Drools	
Endpoint	Provides a consumer endpoint that can implement a service or a router using JBoss Rules.
Namespace	Defines the namespace context used by drools.

ABOUT THE AUTHOR

Jos Dirksen is a software architect for Atos Origin, where he has been the architect for a number of large integration projects over the last couple of years. Jos has worked with various integration products, commercial and open source for the last five years. He co-authored the book *Open Source ESBs in Action* and regularly presents on topics ranging from enterprise integration patterns to JavaFX and OSGi at such conferences as Devoxx and JavaOne.

RECOMMENDED BOOK



Open-Source ESBs in Action describes how to use ESBs in real-world situations. You will learn how the various features of an ESB such as transformation, routing, security, connectivity, and more can be implemented on the example of two open-source ESB implementations: Mule and ServiceMix.

BUY NOW
books.dzone.com/books/opensource-esb

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-65-3
ISBN-10: 1-934238-65-1
50795
9 781934 238653
$7.95

Version 1.0