Introduction to Apache ServiceMix

09/06 Version 1.2.1

Introduction to Apache ServiceMix

Apache ServiceMix is a Java Business Integration (JBI) compliant Enterprise Service Bus (ESB). This document introduces Apache ServiceMix and explains its role as an enterprise service bus within the LogicBlaze FUSE distribution. This document defines some JBI terminology that is necessary to be familiar with before reading the rest of the documentation.

1. Introduction

Apache ServiceMix is an enterprise service bus (ESB) designed according to the Java Business Integration (JBI) specification. An ESB allows disparate applications, platforms and business processes to exchange data with each other in a protocol-neutral way. The JBI specification (JSR 208) defines the manner in which this communication will take place. It is helpful to read the JSR 208 specification to better understand how Apache ServiceMix works.

The diagram below gives a simple view of a JBI container. Application A is an existing or legacy application. It needs to exchange information with an incompatible application, Application B. Apache ServiceMix is represented by the JBI Container. Apache ServiceMix enables the two applications to exchange data.

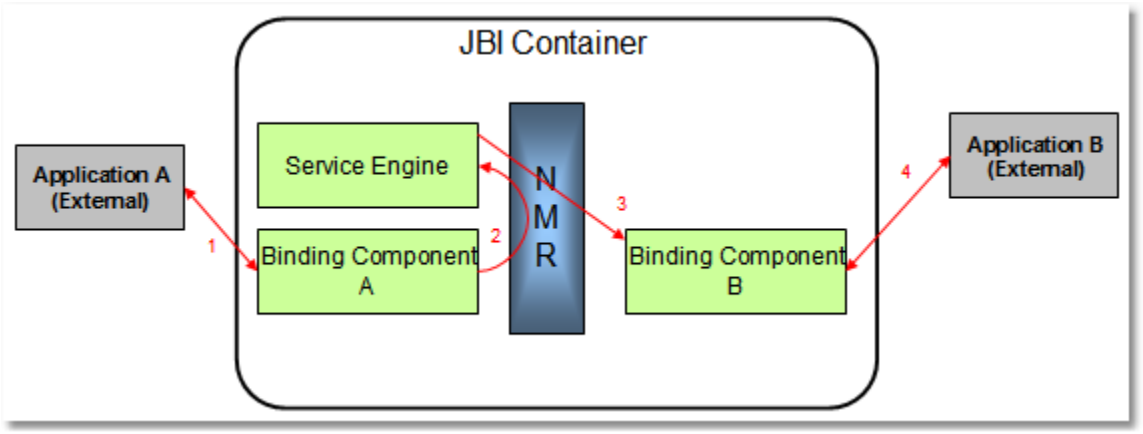

Figure 1: JBI Environment

1. Application A connects into the JBI environment via a binding component (defined later). Binding component A communicates to Application A using a protocol that A understands, e.g., HTTP, JMS, etc.
2. Binding Component A translates the message from its original format into a neutral or normalized format. The normalized message is passed to a service engine component (defined later) for some processing. The message is sent via the Normalized Message Router (NMR).
3. The Service Engine sends the message over the NMR to Binding Component B.
4. Binding Component B will denormalize (convert) the message into a format that Application B can understand and then send the message to Application B.

To get started quickly the next section will define some JBI terminology that will be used throughout the LogicBlaze FUSE documentation.

2. Brief Background on Apache ServiceMix

To understand LogicBlaze FUSE and its capabilities it is critical to have some understanding of Apache ServiceMix. Apache ServiceMix is a JBI compliant ESB. JBI allows third party components to be plugged into a standard infrastructure, and it allows those components to inter-operate in a defined way. There are two major functions of the JBI container that we will focus on here: JBI components and the mechanism for exchanging information.

2.1. JBI Components

The JBI specification requires two types of components. They are defined by JSR 208 as:

> **Service Engine (SE)**: SEs provide business logic and transformation services to other components.
>
> **Binding Component (BCs)**: BCs provide connectivity to services external to the JBI installation. BCs convert protocol and transport specific messages, such as HTTP, SOAP, and JMS messages to a normalized format.

Service engines perform business logic, for example, in the Loan Broker demonstration (see *Loan Broker Tutorial*) there is a service engine that accepts loan rate requests, gets a credit score for a loan applicant, and then determines the lowest loan rate. Service engines also provide transformation services, for example, transforming XML data to an HTML format.

Binding components communicate with external applications in the protocol used by the external application. BCs translate external messages into the normalized format that is used within the JBI infrastructure. All messages passed within the JBI system are in normalized format.

SEs and BCs can be service providers or service consumers or both.

JBI components often function as a container themselves. Artifacts can be deployed to these "container components" to add additional functionality to them. Using the example from JSR 208, "a service engine that provides XSLT-based transformation services would have XSLT style sheets deployed to it, in order to add new transformation operations. This process of adding such component-specific artifacts to an installed component is called *deployment*, to distinguish it from component *installation*. JBI terms a collection of deployment artifacts and associated metadata a *service assembly*." Therefore, components are **installed** on LogicBlaze FUSE; artifacts are **deployed** to a specific component.

Another term you will see is *service unit*. A service unit (SU) is a single deployment artifact to be deployed on a component. A Service Assembly (SA) is a standard packaging of artifacts to be deployed as a unit, which provides composite deployment capability. The standard packaging of SAs and SUs is essentially a predefined directory structure of files put together in a jar file. The details of this are discussed in JSR 208.

There is a third type of component that is not required by the JBI specification. It is specific to Apache ServiceMix and is called a lightweight component. A lightweight component is an easy-to-write POJO (Plain Old Java Object) that activates a single JBI endpoint and does not support service unit deployments.

Standard JBI components can be deployed at run-time and accept service unit deployments. `servicemix-lwcontainer` is an Apache ServiceMix standard component, in other words it is not a lightweight component. Lightweight components can be deployed on it. By deploying lightweight components on `servicemix-lwcontainer` they can be added or removed at run-time, the same as a standard JBI component.

JSR 208 defines JBI endpoints as, "Endpoints, in WSDL 2.0, refer to an address, accessible by a particular protocol, used to access a particular service".

2.2. Message Exchange within the JBI Environment

Messages are exchanged over the normalized message router (NMR) in the JBI environment. Messages sent via the NMR are normalized messages, which means they have been transformed into a neutral format that all the components understand.

JBI implementations are required to support four standard message exchange sequences called message exchange patterns (MEPs). A component can play two roles with respect to JBI message exchanges. In a given exchange, the originator component is said to be the **consumer** and the target component is said to be the **provider**.

The JBI specification defines service invocation as an instance of an interaction between a service consumer and a service provider. As stated in JSR 208, the following four service invocation patterns are required to be supported by any JBI implementation:

Table 1: *Message Exchange*

Message Exchange Pattern	Direction	Description
In-Only	in	**One-Way**: consumer issues a request to a provider with no error (fault) path provided.
Robust In-Only	in	**Reliable One-Way**: consumer issues a request to a provider. Provider may respond with a fault if it fails to process the request.
In-Out	in and out	**Request-Response**: consumer issues a request to a provider, with expectation of response. Provider may respond with a fault if it fails to process the request.
In, Optional Out	in and out	**Request Optional-Response**: consumer issues a request to provider, which may result in a response. Consumer and provider both have the option of generating a fault in response to a message received during the interaction.

The WSDL 2.0 Extensions specification defines a **Message Exchange Pattern** (MEP) as: "the sequence and cardinality of abstract messages listed in an operation". JBI uses this concept to define interactions between two nodes: the consumer node and the provider node. The pattern is defined in terms of message type (normal or fault), and message direction.

MEPs always reflect the provider's point of view. For example, in a request-response interaction, the MEP is in-out, which reflects the flow of messages as seen by the provider. From the consumer's perspective, the direction of message flow is reversed, but the MEP used by the NMR in its interactions with the consumer will always reflect the provider's perspective. This is a conventional practice when dealing with WSDL MEPs.

Looking back at Figure 1, the message exchange patterns are:

1. Binding Component A is a service provider and the MEP is In-Only.
2. Service Engine is a service provider and the MEP is In-Only.
3. Binding Component B is a service provider and the MEP is In-Only.

3. Apache ServiceMix Components

Apache ServiceMix comes with many pre-built components. Below is a matrix of ServiceMix components and a brief description of them.

Table 2: *Apache ServiceMix Components*

Component Name	Type	Description	Role	MEP
Cache	LW SE	Used for caching service invocations to avoid unnecessary load on expensive services.	Mixed	In-Out
ChainedComponent	LW SE	Router - Will route a message from one component to another.	Mixed	In-Out
EchoComponent	LW SE	Echoes back what it receives	Provider	In-Out
Drools	LW SE		Mixed	In-Only
Email	LW BC	Email support via JavaMail (http://java.sun.com/products/javamail/) http://servicemix.codehaus.org/maven/servicemix-components/apidocs/org/apache/servicemix/components/email/MimeMailSender.html	Provider	In-Only
		http://servicemix.codehaus.org/maven/servicemix-components/apidocs/org/apache/servicemix/components/email/SimpleMailSender.html	Provider	In-Only
		http://servicemix.codehaus.org/maven/servicemix-components/apidocs/org/apache/servicemix/components/email/MimeMailPoller.html	Consumer	In-Only
File	LW BC	Components for writing messages to files and polling directories and sending files into the JBI. http://servicemix.codehaus.org/maven/servicemix-components/apidocs/org/apache/servicemix/components/file/FileSender.html	Provider	In-Only
		http://servicemix.codehaus.org/maven/servicemix-components/apidocs/org/apache/servicemix/components/file/FilePoller.html	Consumer	In-Only

Component Name	Type	Description	Role	MEP
FTP	LW BC	FTP support via the Jakarta Commons Net library. http://servicemix.codehaus.org/maven/servicemix-components/apidocs/org/apache/servicemix/components/net/FTPPoller.html	Provider	In-Only
			Consumer	In-Only
Groovy	LW SE	This component allows Groovy scripts to be used as endpoints, transformers, or services. This allows you to combine the power of the Groovy scripting language with the ServiceMix JBI container.	Provider	In-Only or In-Out depending on the Groovy script sent to the Groovy SE.
HTTP	LW BC	Both client-side GET/POST with commons `httpclient` and server side processing with Servlets or Jetty.		
		`HttpConnector`	Consumer	In-Out
		`HttpInOnlyBinding`	Consumer	In-Only
		`HttpSoapConnector`	Consumer	In-Out
		`HttpInOnlySoapBinding`	Consumer	In-Only
Jabber	LW BC	Provides bindings to Jabber network via the Extensible Messaging and Presence Protocol (XMPP) protocol.		
		`JabberReceiver`	Consumer	In-Only
		`JabberChatSender`	Provider	In-Only
		`JabberGroupChatSender`	Provider	In-Only
JCA	LW SE	Allows the Java Connector Architecture to be used for efficient thread pooling, transaction handling and consumption on JMS or other Resource Adapters.	Consumer	In-Only
JMS	LW BC	JMS via the Java Messaging Service plus all of the great, reliable, and scalable transports in ActiveMQ which includes persistence, recovery, and transaction support.		
		`JmsInBinding`	Consumer	In-Only
		`JmsInUsingJCABinding`	Consumer	In-Only
		`JmsReceiverComponent`	Consumer	In-Out
		`JmsSenderComponent`	Provider	In-Only

Component Name	Type	Description	Role	MEP
		`JmsServiceComponent`	Provider	In-Out
MockServiceComponent	LW SE	Useful for controlling what is sent back to consumer. You have to configure the answer. Sends a pre-configured response back.	Provider	In-Out
PipelineComponent	LW SE	Will Bridge an In-Only request to an In-Out request exchange pattern.	Mixed	In-Out
Quartz	LW SE	A component for job scheduling.	Consumer	In-Only
Reflection	LW SE	The reflection API represents, or reflects, the classes, interfaces, and objects in the current Java Virtual Machine. This API is handy if you are writing development tools such as debuggers, class browsers, and GUI builders.		
		`o.a.s.components.reflection.proxyInOnlyBinding`[1]	Consumer	In-Only
		`o.a.s.components.relection.proxyInOutBinding`	Consumer	In-Out
RSS	LW BC	Support via Rome library for accessing and processing RSS feeds.		
		`Rsspollingcomponent`	Consumer	In-Only
		`FeedWriter`	Provider	In-Only
SAAJ	LW BC	SAAJ is for Soap With Attachments and Apache Axis support.	Provider	In-Out
Scripting	LW SE	A component to allow any JSR 223 compliant scripting engine to be used to easily create a component, perform a transformation, or be an expression language.	Provider	In-Only or In-Out depending on the Groovy script sent to the Groovy SE.
servicemix-bpe	JBI SE	WSDL 2.0 Adjuncts defines pre-defined extensions for WSDL 2.0, including MEPs, operation styles, and binding extensions.	Provider Consumer	All MEPs supported.
servicemix-eip	JBI SE	A routing container where different routing patterns can be deployed as service unit. Based on the EIP Patterns book.	Consumer and Provider	n/a

1 `o.a.s = org.apache.servicemix.`

Component Name	Type	Description	Role	MEP
servicemix-http	JBI BC	HTTP binding.	Consumer and Provider	In-Only / In-Out
servicemix-jms	JBI BC	JMS binding.	Consumer and Provider	In-Only / In-Out
servicemix-jsr181	JBI SE	Hosts annotated POJOs.	Provider	In-Only / In-Out
servicemix-lwcontainer	JBI SE	Acts as a host for lightweight components.	n/a	n/a
servicemix-wsn2005	JBI SE	WS-Notification.	Consumer and Provider	n/a
StreamWriterComponent	LW SE	Similar to TraceComponent, writes content of input message to the stream requested.	Provider	In-Only
TraceComponent	LW SE	Logs output to console	Provider	In-Only
Validation	LW SE	For schema validation of documents using Java API for XML Processing (JAXP 1.3) and XMLSchema (http://www.w3.org/XML/Schema) or RelaxNG - a schema language for XML (http://relaxng.org/). If the inputted XML is validated as okay, the same XML is sent back. If there is an error a fault is returned to the sender.	Mixed	As the Provider, the MEP is In-Only, as the consumer it is also In-Only. This component can also have an In-Out exchange pattern, which is the main pattern for this component
VFS	LW BC	VFS via the Jakarta Commons Net library which provides access to file systems, jars/zips/bzip2, temporary files, WebDAV, Samba (CIFS), HTTP, HTTPS, FTP and SFTP among others.		
		`FilePoller`	Consumer	In-Only
		`FileWriter`	Provider	In-Only
WSIF	LW BC	WSIF for integration with the Apache Web Service Invocation Framework (WSIF).	Provider	In-Out or In-Only
Xfire	LW BC	XFire SOAP stack	Provider	In-Out

Component Name	Type	Description	Role	MEP
XPath Routing	LW SE	Used to perform content based routing in an ESB. This means you route messages around your service bus based on the message properties or the content of the messages. When integrating systems across language boundaries its common to use XML as a universal message format; so XPath is an ideal tool to perform content based routing and transformation.	Mixed	Receives an In-Only exchange and sends an In-Only message to another component.
XSLT	LW SE	The `XsltComponent` will perform an XSLT transformation of an inbound Normalized Message and generate an output message as a Normalized Message.	Mixed	As the Provider, the MEP is In-Only, as the consumer it is also In-Only. This component can also have an In-Out exchange pattern, which is the main pattern for this component.

4. Additional Resources

For more information please see the Apache ServiceMix Wiki documentation: http://incubator.apache.org/servicemix/.

To read the Java Business Integration specification (JSR 208) please see: http://www.jcp.org/en/jsr/detail?id=208.

For more information on WSDL 2.0 please see: http://www.w3.org/TR/2005/WD-wsdl20-adjuncts-20050803.

For more information on WSDL 2.0 Adjuncts please see: http://www.w3.org/TR/wsdl20-adjuncts/.

For more information on Groovy please visit: http://groovy.codehaus.org/.

For more information on Drools please visit: http://docs.codehaus.org/display/DROOLS/Home.

For more information on Jabber please visit: http://www.jabber.org.

To see examples of service assemblies and a demo of deployment, please read the *Loan Broker Tutorial*.