Red Hat JBoss Fuse 6.0

Getting Started with JBoss Fuse 6.0

Sameer Parulkar, JBoss Product Marketing Manager
Kenneth Peeples, JBoss Technology Evangelist

April 16, 2013

JBoss Fuse Getting Started

- Part 1 – JBoss Fuse Overview
- Part 2 - Demonstration of a JBoss Fuse Enterprise Integration Pattern Quickstart

Part 1: JBoss Fuse

Overview

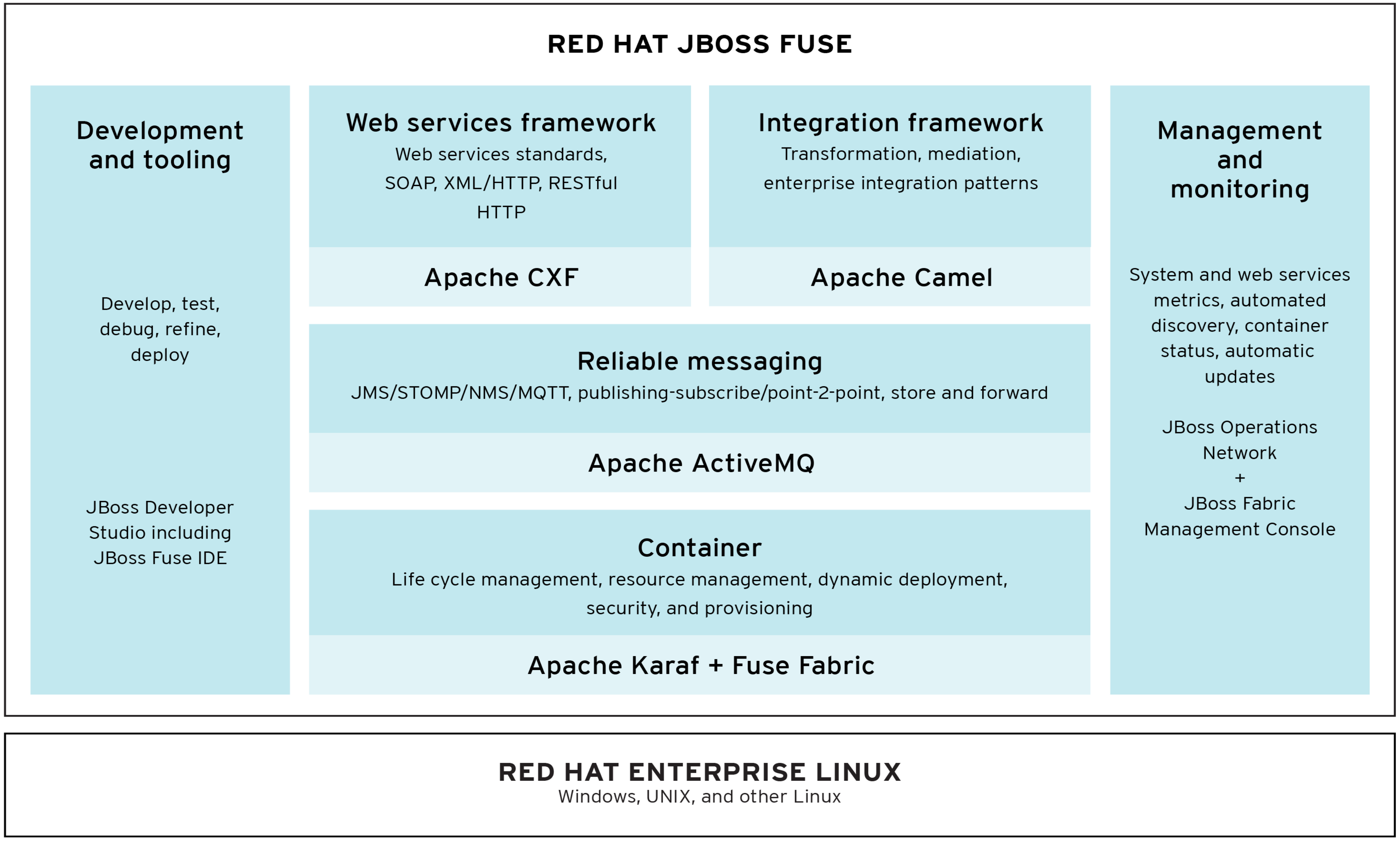
JBoss Fuse Overview

- Product Launch for JBoss Fuse
- Features and Benefits – JBoss Fuse

JBoss Fuse Launch Overview

- JBoss Fuse - a small-footprint, flexible open source Enterprise Service Bus (ESB)
  - Rebranding/Repackaging of Fuse ESB Enterprise
- Includes JBoss A-MQ – a small-footprint, high performance messaging platform
  - Rebranding/Repackaging of Fuse MQ Enterprise
- GA for JBoss Fuse 6.0 and JBoss A-MQ 6.0 announced on Apr 11, 2013
- Available in community for development use – Try our products!
  - JBoss Fuse – http://jboss.org/products/fuse
  - JBoss A-MQ – http://jboss.org/products/amq

JBoss Fuse – Integration Everywhere!


RED HAT JBOSS FUSE
Development and tooling
Develop, test, debug, refine, deploy
JBoss Developer Studio including JBoss Fuse IDE
Web services framework
Web services standards, SOAP, XML/HTTP, RESTful HTTP
Apache CXF
Integration framework
Transformation, mediation, enterprise integration patterns
Apache Camel
Reliable messaging
JMS/STOMP/NMS/MQTT, publishing-subscribe/point-2-point, store and forward
Apache ActiveMQ
Container
Life cycle management, resource management, dynamic deployment, security, and provisioning
Apache Karaf + Fuse Fabric
Management and monitoring
System and web services metrics, automated discovery, container status, automatic updates
JBoss Operations Network + JBoss Fabric Management Console
RED HAT ENTERPRISE LINUX
Windows, UNIX, and other Linux

JB0012

JBoss Fuse Benefits

No license fees – Try before you buy, and deploy widely without incurring exorbitant costs

Faster time to solution – Includes Apache Camel®, the easiest way to rapidly integrate your enterprise assets

Extensive connectivity – includes connectivity for the most common tasks

Flexible configuration – Dynamic configurations enable customized solutions at every endpoint, distributor, outlet or device.

Proven reliability and Support – track record of supporting mission-critical applications and backed by Red Hat's award winning support

Part 2: JBoss Fuse

Demonstration of a JBoss Fuse Enterprise Integration Pattern Quickstart

Topics Covered during the demonstration

- How to define a Camel route using the Blueprint XML syntax
- How to build and deploy a Fuse Application Bundle (FAB)
- How to combine multiple Enterprise Integration Patterns (Wiretap, Splitter, Recipient List and Filter) to create an integration solution
- How to define and use a bean to process a message
- How to use a 'direct:' endpoint to link multiple smaller routes together

Scenario for Demonstration

1) First we want to make sure we retain a copy of the original orders file containing several orders for zoos around the world we received. This is done using the Wiretap EIP.
2) After saving the original, we want to split the file up into the individual orders. This is done using the Splitter EIP.
3) Then we want to store the orders in separate directories by geographical region. This is done using a Recipient List EIP.
4) Finally, we want to filter out the orders that container more than 100 animals and generate a message for the strategic account team. This is done using a Filter EIP.

Source Files used in the demonstration

Source files in the fuse-eip-quickstart project -

- pom.xml - the Maven POM file for building the example

In src/main -

- java/org.jboss.fuse.examples/eip/RegionSupport.java - a Java class used to determine the region code used by the recipient list
- resources/OSGI-INF/blueprint/eip.xml - the OSGI Blueprint file that defines the routes

In src/test -

- data/orders.xml - the data file that can be used to test the route
- java/RegionSupportTest.java - a JUnit test class for RegionSupport

What are Enterprise Integration Patterns (EIP)?

Gregor Hohpe and Bobby Woolfe's book, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, contains a set of sixty five Enterprise Integration Patterns (EIPs).

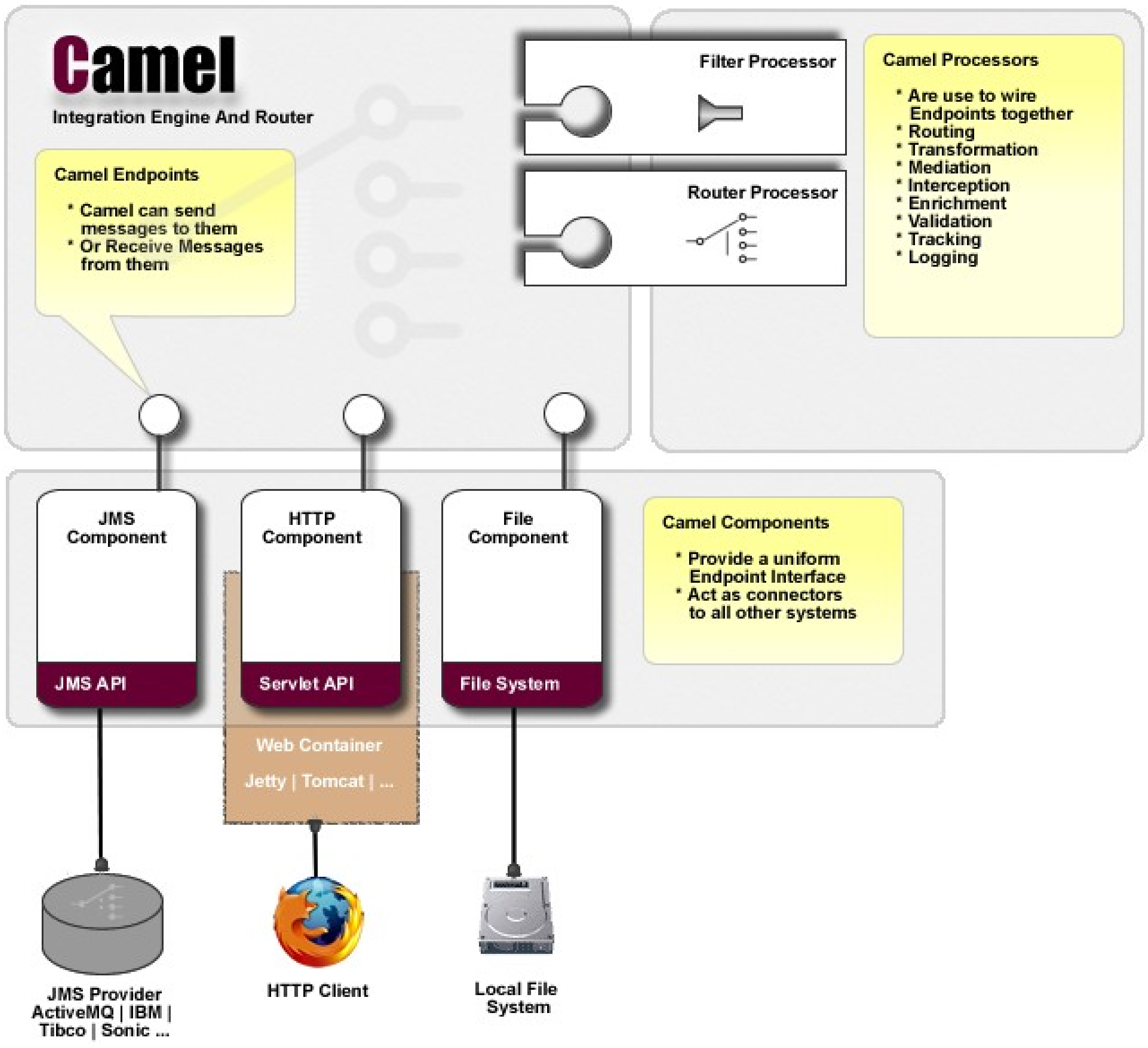
Apache Camel is an open source Java framework that focuses on making integration easier and more accessible to developers. Camel provides:

- implementations of the widely used EIPs
- connectivity to multiple transports and APIs
- easy to use Domain Specific Language (DSL) to wire EIPs and transports together

More information:
http://camel.apache.org/enterprise-integration-patterns.html

Camel Architecture

DSL used to wire endpoints and processors together to form routing rules



What is Camel DSL?

Camel uses a Java Domain Specific Language or DSL for creating Enterprise Integration Patterns or Routes in a variety of domain-specific languages (DSL) as listed below.

- Java DSL - A Java based DSL using the fluent builder style.
- Spring XML - A XML based DSL in Spring XML files
- Blueprint XML - A XML based DSL in OSGi Blueprint XML files
- Groovy DSL - A Groovy based DSL using Groovy programming language
- Scala DSL - A Scala based DSL using Scala programming language
- Annotation DSL - Use annotations in Java beans.

More Information: http://camel.apache.org/dsl.html

What are Camel URI Components?

http://camel.apache.org/file2.html - The File component provides access to file systems, allowing files to be processed by any other Camel Components or messages from other components to be saved to disk

http://camel.apache.org/direct.html - The direct: component provides direct, synchronous invocation of any consumers when a producer sends a message exchange. This endpoint can be used to connect existing routes in the same camel context.

More information:
http://camel.apache.org/components.html

OSGI Blueprint File – eip.xml

<project_home>/projects/fuse-eip-quickstart/src/main/resources/OSGI-INF/blueprint/eip.xml - OSGI Blueprint file that defines the routes for our quickstart

This is the OSGi Blueprint XML file defining the Camel context and routes. Because the file is in the OSGI-INF/blueprint directory inside our JAR, it will be automatically activated as soon as the artifact is installed. The root element for any OSGi Blueprint file is 'blueprint' - you also see the namespace definitions for both the Blueprint and the Camel namespaces.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="
             http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
             http://camel.apache.org/schema/blueprint
http://camel.apache.org/schema/blueprint/camel-blueprint.xsd">
```

OSGI Blueprint File - Continued

We are using the OSGi Blueprint XML syntax to define a bean that we use in our Camel route to determine the geographical region. The code for this bean can be found in src/main/java/org.jboss.fuse.examples/eip/RegionSupport.java

```
<bean id="MyRegionSupport"
class="org.jboss.fuse.examples.eip.RegionSupport" />
```

The namespace for the camelContext element in Blueprint is 'http://camel.apache.org/schema/blueprint'. We can also define namespace prefixes we want to use in the XPath expressions in our CBR here. We defined the namespace prefix order so that we use the namespace from the order messages in the XPath expressions. While it is not required to assign id's to the <camelContext/> and <route/> elements, it is a good idea to set them for runtime management purposes (logging, JMX MBeans, ...)

```
 <camelContext xmlns="http://camel.apache.org/schema/blueprint"
                 xmlns:order="http://fusesource.com/examples/order/v7"
                 id="eip-example-context">
```

OSGI Blueprint File - Continued

This is the main Camel route. In Camel, you can split up routes into more manageable bits and pieces and using a direct: endpoint to link those routes together again. In this case, files will be sent to the direct:splitter endpoint after a wiretap has sent a copy of the message to the direct:wiretap endpoint. The routes behind these endpoints are defined below.

```
  <route id="mainRoute">
      <from uri="file:work/eip/input" />
      <log message="[main]    Processing ${file:name}" />
      <wireTap uri="direct:wiretap" />
      <to uri="direct:splitter" />
      <log message="[main]    Done processing ${file:name}" />
  </route>
```

This route starts with the direct:wiretap endpoint we used in our main route. Whenever the wiretap in the main route sends a message to this endpoint, the route below will log a human-friendly message and store a copy of the message in the work/eip/archive directory.

```
    <route id="wiretapRoute">
      <from uri="direct:wiretap" />
      <log message="[wiretap]  Archiving ${file:name}" />
      <to uri="file:work/eip/archive" />
    </route>
```

OSGI Blueprint File - Continued

This route starts with the direct:splitter endpoint we used in our main route. In this route, we will split a file containing multiple <order/> elements into separate messages, so we can store the orders in a separate directory for every geographical region.

```
    <route id="splitterRoute">
         <from uri="direct:splitter"/>
         <split>
```

We are using XPath to split the message body. The namespace prefix, order, used in the XPath expression was defined on the camelContext element.

```
         <xpath>//order:order</xpath>
```

Headers can be used to add meta-data to the message without altering the actual message body. In this case, we want to keep the order XML message, but we add two additional pieces of information:

- the order id : we use XPath to determine the order id
- the geographical region : we are using a method called getRegion() on the MyRegionSupport bean we defined earlier

```
            <setHeader headerName="orderId">
               <xpath>/order:order/@id</xpath>
            </setHeader>
            <setHeader headerName="region">
               <method bean="MyRegionSupport" method="getRegion" />
            </setHeader>
```

OSGI Blueprint File - Continued

Another human-friendly log message, this time using the extra headers we just added to the message

```
        <log message="[splitter] Shipping order ${header.orderId} to region ${header.region}"/>
```

A recipient list can be used to dynamically determine the next steps for a message. In this case, we dynamically generate the file: endpoint uri using the two headers we defined earlier. We will also send the message to the direct:filter endpoint to see if the order needs attention from our strategic account management team.

```
        <recipientList>
          <simple>file:work/eip/output/${header.region}?fileName=${header.orderId}.xml,direct:filter</simple>
        </recipientList>
      </split>
    </route>
```

OSGI Blueprint File - Continued

This route starts with the direct:filter endpoint we used in the <recipientList> in our splitter route. It uses a Filter EIP to filter out the order messages that contain more than 100 animals in a single order. For all messages that meet the XPath filter expression, an extra message will appear in the logs.

```
        <route id="filterRoute">
            <from uri="direct:filter" />
            <filter>
                <xpath>sum(//order:quantity/text()) > 100</xpath>
                <log message="[filter]   Order ${header.orderId} is an order for more than 100 animals" />
            </filter>
        </route>
</camelContext>
```

What is a Fuse Application Bundle (FAB)?

Fuse Bundles are designed to make it very easy for application developers to create applications that can be deployed in OSGi containers or web containers so that its:

- easy to deploy and control which classes are shared across deployment units
- there's a common model of class loaders across your build tool (e.g. Maven), your IDE (e.g. with Maven integration) and container with few surprises
- developers can focus on getting things done rather than fighting OSGi metadata or struggling with OSGi specific metadata generation tools.

A Fuse Bundle, or FAB, is any jar created using Apache Maven or similar build tools so that inside the jar there is a pom.xml file at META-INF/maven/groupId/artifactId/pom.xml (and pom.properties file) which contain the transitive dependency information for the jar.

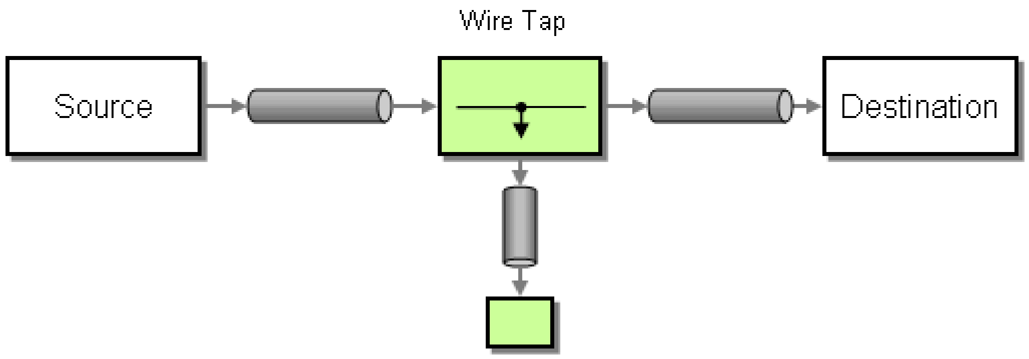
In your OSGi container such as Fuse ESB or Apache Karaf with the Fuse bundle feature installed - you can then install any FAB as if it were a WAR or Karaf feature.

To install a FAB with some value of groupId, artfactId, version type the following command into the console of either Fuse ESB or Apache Karaf (if it has the Fuse bundle feature installed)

install fab:mvn:groupId/artifactId/version

For More Information: http://fuse.fusesource.org/bundle/overview.html

Wiretap - System Management EIP

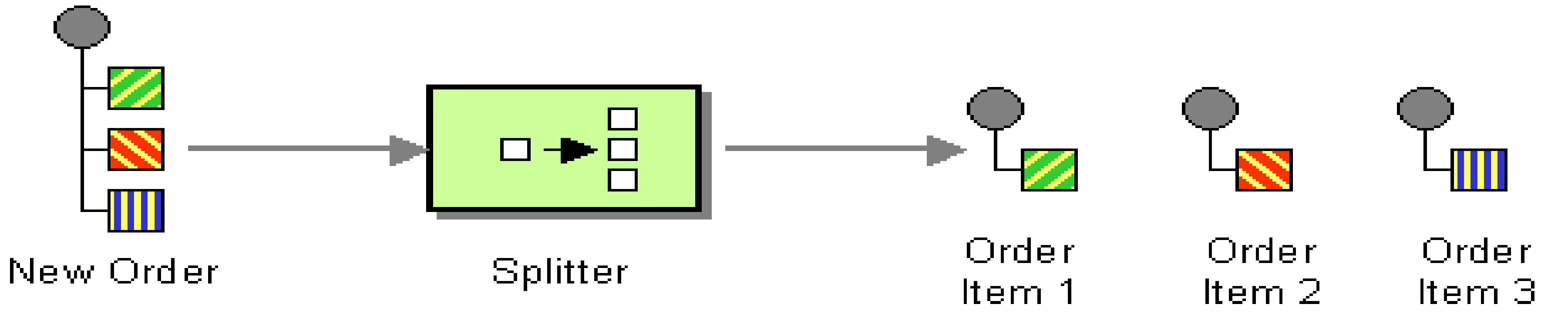


Wire Tap, from the EIP patterns, allows you to route messages to a separate location while they are being forwarded to the ultimate destination

```
<route id="mainRoute">
  <from uri="file:work/eip/input" />
  <log message="[main]    Processing ${file:name}" />
  <wireTap uri="direct:wiretap" />
  <to uri="direct:splitter" />
  <log message="[main]    Done processing ${file:name}" />
</route>
```

```
<route id="wiretapRoute">
  <from uri="direct:wiretap" />
  <log message="[wiretap]  Archiving $
{file:name}" />
  <to uri="file:work/eip/archive" />
</route>
```

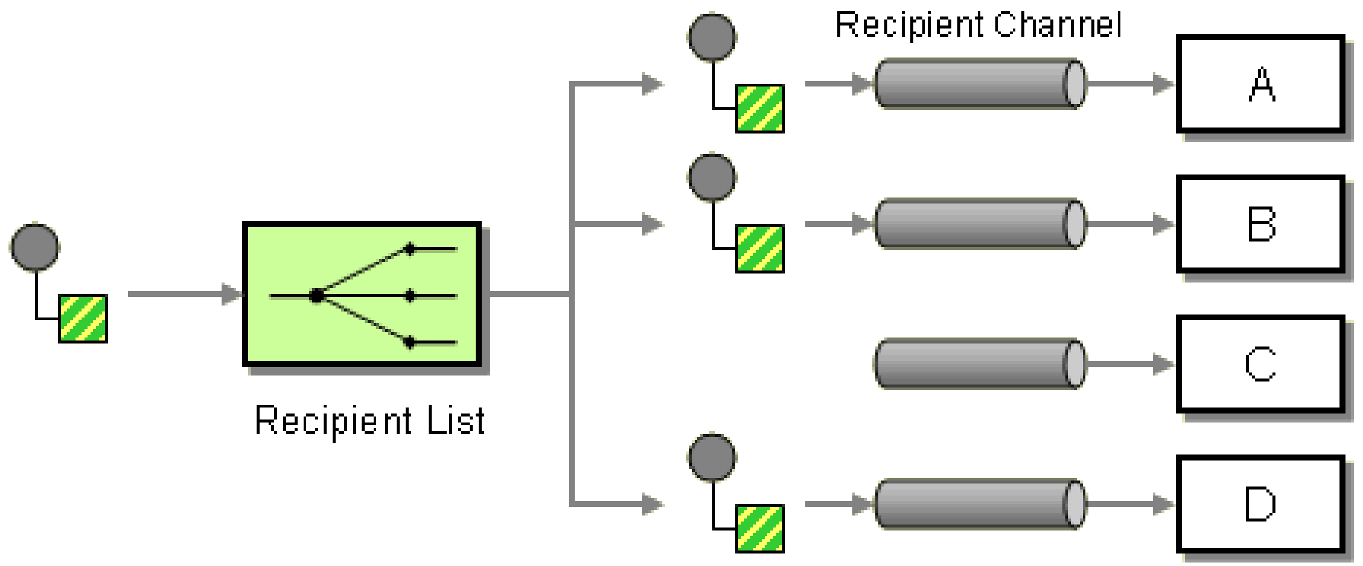
Splitter - Message Routing EIP



The Splitter, from the EIP patterns, allows you split a message into a number of pieces and process them individually

```
 <route id="splitterRoute">
   <from uri="direct:splitter"/>
   <split>
      <xpath>//order:order</xpath>
      <setHeader headerName="orderId">
         <xpath>/order:order/@id</xpath>
      </setHeader>
      <setHeader headerName="region">
         <method bean="MyRegionSupport" method="getRegion" />
      </setHeader>
      <log message="[splitter] Shipping order ${header.orderId} to region ${header.region}"/>
      <recipientList>
         <simple>file:work/eip/output/${header.region}?fileName=${header.orderId}.xml,direct:filter</simple>
      </recipientList>
   </split>
</route>
```

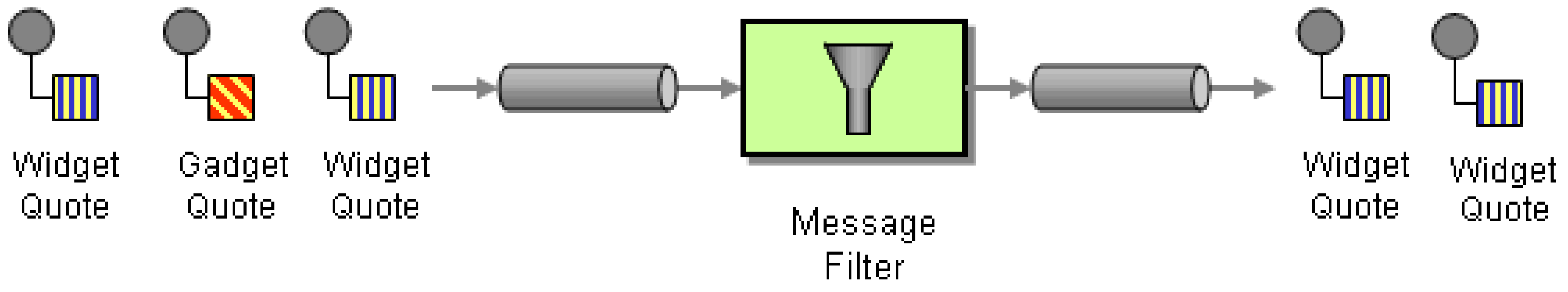
Recipient List - Message Routing EIP



The Recipient List, from the EIP patterns, allows you to route messages to a number of dynamically specified recipients.

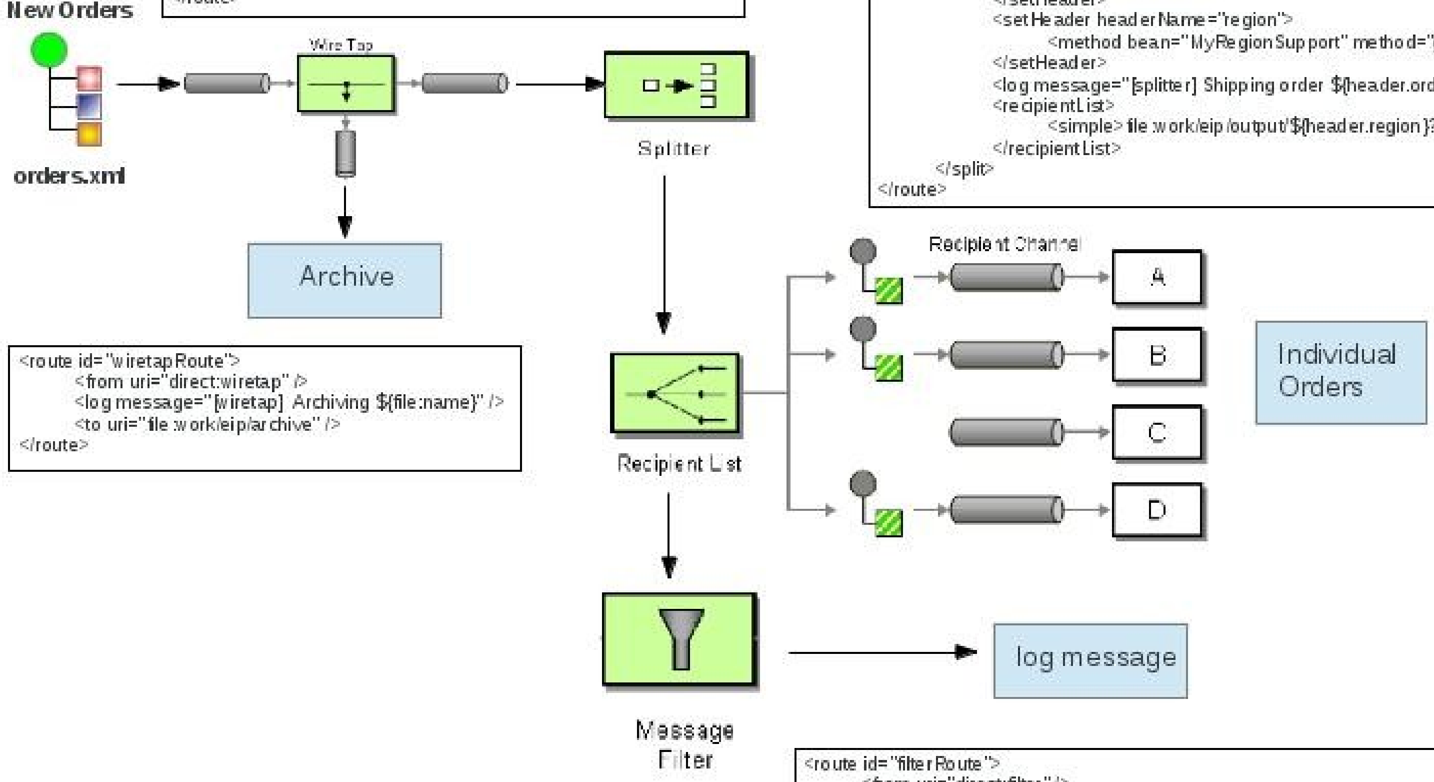
```
<recipientList>
   <simple>file:work/eip/output/${header.region}?fileName=${header.orderId}.xml,direct:filter</simple>
</recipientList>
```

Message Filter - Message Routing EIP



The Message Filter, from the EIP patterns, allows you to filter messages.

```
  <route id="filterRoute">
      <from uri="direct:filter" />
      <filter>
          <xpath>sum(//order:quantity/text()) > 100</xpath>
          <log message="[filter]   Order ${header.orderId} is an order for more than 100
animals" />
      </filter>
  </route>
```



```
<route id="mainRoute">
        <from uri="file:work/eip/input" />
        <log message="[main]    Processing ${file:name}" />
        <wireTap uri="direct:wiretap" />
        <to uri="direct:splitter" />
        <log message="[main]    Done processing ${file:name}" />
</route>
```

```
<route id="splitterRoute">
        <from uri="direct:splitter"/>
        <split>
                <xpath>//order:order</xpath>
                <setHeader headerName="orderId">
                <xpath>/order:order/@id</xpath>
                </setHeader>
                <setHeader headerName="region">
                        <method bean="MyRegionSupport" method="getRegion" />
                </setHeader>
                <log message="[splitter] Shipping order ${header.orderId} to region ${header.region}"/>
                <recipientList>
                        <simple>file:work/eip/output/${header.region}?fileName=${header.orderId}.xml,direct:filter</simple>
                </recipientList>
        </split>
</route>
```

```
<route id="wiretapRoute">
        <from uri="direct:wiretap" />
        <log message="[wiretap]  Archiving ${file:name}" />
        <to uri="file:work/eip/archive" />
</route>
```

```
<route id="filterRoute">
        <from uri="direct:filter" />
        <filter>
        <xpath>sum(//order:quantity/text()) > 100</xpath>
        <log message="[filter]   Order ${header.orderId} is an order for more than 100 
animals" />
        </filter>
</route>
```

Prerequisites

Maven 3.0.3 or higher

JDK 1.7

JBoss Fuse 6

JBoss Developer Studio 6

Source

https://github.com/kpeeples/fuse-eip-quickstart.git

Downloads

http://www.jboss.org/products
(original quickstart under examples/eip)

Demonstration Steps

Step 1 – Install and Configure product used.

- View the README in 'installs' directory.
- Download the JBoss Fuse product from the Red Hat Customer Support Portal or JBoss.org
- Add to the installs directory.
- Update the zip filename in init.sh in the FUSE_BIN variable.

Step 2 - Run 'init.sh'

- From the command prompt run the init script, ./init.sh, from the command line
- View output for build success
- sample posted to init.sh.output under the support folder
- View the eip-6.0.0.redhat-024.jar in the maven repository

Step 3 - Install and/or Setup JBDS for project import

- Download JBDS
- Create workspace /home/kpeeples/workspaces/fuse-eip-quickstart

Step 4 - Add the JBoss Fuse server

- admin=redhat,admin update users.properties in /etc first
- Run the new server wizard
- Select Red Hat JBoss Fuse 6.x Server
- Select installed product under target folder
- Enter admin/redhat from above

Documentation Steps - Continued

Step 5 - Start JBoss Fuse server and the shell should display

```
[kpeeples@localhost bin]$ ./fuse Please wait while JBoss Fuse is loading... 100%
[=================================================================]
JBoss Fuse (6.0.0.redhat-015)
http://www.redhat.com/products/jbossenterprisemiddleware/fuse/
Hit '' for a list of available commands and '[cmd] --help' for help on a specific command.
Hit '' or 'osgi:shutdown' to shutdown JBoss Fuse.
JBossFuse:karaf@root>
```

Step 6 - Import the existing maven project to review the files

Step 7 - Install the Fuse Application Bundle (FAB)

- Run osgi:install -s fab:mvn:org.jboss.fuse.examples/eip/6.0.0.redhat-024
- Response should be the Bundle ID

Step 8 - View the OSGi list to make sure the FAB has been created and active

- Run osgi:list - l

-
  ```
  [ 234] [Active     ] [Created     ] [       ] [   60]
  fab:mvn:org.jboss.fuse.examples/eip/6.0.0.redhat-024
  ```

Demonstration Steps

Step 9 - As soon as the Camel route has been started, you will see a directory `work/eip/input` in your JBoss Fuse installation.

Step 10 - Copy the file you find in this example's `src/test/data` directory to the newly created `work/eip/input` directory.

Step 11 - Wait a few moment and you will find multiple files organized by geographical region under `work/eip/output':

** `2012_0003.xml` and `2012_0005.xml` in `work/eip/output/AMER`
** `2012_0020.xml` in `work/eip/output/APAC`
** `2012_0001.xml`, `2012_0002.xml` and `2012_0004.xml` in `work/eip/output/EMEA`

Step 12 - Use `log:display` on the ESB shell to check out the business logging.

```
[main]    Processing orders.xml
[wiretap]  Archiving orders.xml
[splitter] Shipping order 2012_0001 to region EMEA
[splitter] Shipping order 2012_0002 to region EMEA
[filter]   Order 2012_0002 is an order for more than 100 animals
```

Log Display

```
18:35:50,229 | INFO  | //work/eip/input | mainRoute                        | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [main]    Processing orders.xml
18:35:50,235 | INFO  | ead #2 - WireTap | wiretapRoute                     | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [wiretap]  Archiving orders.xml
18:35:50,268 | INFO  | //work/eip/input | splitterRoute                    | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [splitter] Shipping order 2012_0001 to region EMEA
18:35:50,325 | INFO  | //work/eip/input | splitterRoute                    | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [splitter] Shipping order 2012_0002 to region EMEA
18:35:50,369 | INFO  | //work/eip/input | filterRoute                      | 130 - org.apache.camel.camel-core
- 2.10.0.redhat-60024 | [filter]   Order 2012_0002 is an order for more than 100 animals
18:35:50,373 | INFO  | //work/eip/input | splitterRoute                    | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [splitter] Shipping order 2012_0003 to region AMER
18:35:50,412 | INFO  | //work/eip/input | filterRoute                      | 130 - org.apache.camel.camel-core
- 2.10.0.redhat-60024 | [filter]   Order 2012_0003 is an order for more than 100 animals
18:35:50,415 | INFO  | //work/eip/input | splitterRoute                    | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [splitter] Shipping order 2012_0004 to region EMEA
18:35:50,456 | INFO  | //work/eip/input | splitterRoute                    | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [splitter] Shipping order 2012_0005 to region AMER
18:35:50,497 | INFO  | //work/eip/input | splitterRoute                    | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [splitter] Shipping order 2012_0020 to region APAC
18:35:50,539 | INFO  | //work/eip/input | mainRoute                        | 130 - org.apache.camel.camel-
core - 2.10.0.redhat-60024 | [main]    Done processing orders.xml
```

CamelOne 2013
The Open Source Integration Conference
JUNE 10-11 • BOSTON, MA
▶ Register
Learn more: www.camelone.com

References

http://camel.apache.org/components.html

http://camel.apache.org/enterprise-integration-patterns.html
http://camel.apache.org/recipient-list.html for the Recipient List EIP
http://camel.apache.org/wire-tap.html for the Wire Tap EIP
http://camel.apache.org/message-filter.html for the Message Filter EIP
http://camel.apache.org/splitter.html for the Splitter EIP

Contact Information

Sameer Parulkar
Product Marketing Manager
sparulka@redhat.com

Kenneth Peeples
JBoss Technology Evangelist
kpeeples@redhat.com

Cleanup to initialize demo

- Remove target in fuse-eip-quickstart
- Remove target in project subdirectory in fuse-eip-quickstart
- Remove Server instance
- Remove workspace
- Remove jar from repo