

# Appendix A. Migrating from ServiceMix EIP

*If you are currently an Apache ServiceMix user, you might already have implemented some Enterprise Integration Patterns using the ServiceMix EIP module. It is recommended that you migrate these legacy patterns to FUSE Mediation Router, which has more extensive support for Enterprise Integration Patterns. After migrating, you can deploy your patterns either into a FUSE ESB container or into a ServiceMix container.*

Migrating Endpoints .....	112
Common Elements .....	115
ServiceMix EIP Patterns .....	117
Content-Based Router .....	119
Content Enricher .....	121
Message Filter .....	123
Pipeline .....	125
Resequencer .....	127
Static Recipient List .....	129
Static Routing Slip .....	131
Wire Tap .....	132
XPath Splitter .....	134

# Migrating Endpoints

---

## Overview

A typical ServiceMix EIP route exposes a service that consumes exchanges from the NMR. The route also defines one or more target destinations, to which exchanges are sent. In the Mediation Router environment, the exposed ServiceMix service maps to a *source endpoint* and the ServiceMix target destinations map to *target endpoints*. The Mediation Router source endpoints and target endpoints are both defined using *endpoint URIs* (see [Architecture](#) in the *Getting Started*).

When migrating endpoints from ServiceMix EIP to FUSE Mediation Router, you will need to express the ServiceMix services/endpoints as Mediation Router endpoint URIs. You can adopt one of the following approaches:

- Connect to an existing ServiceMix service/endpoint through the ServiceMix Camel module (which integrates Mediation Router with the NMR).
- Alternatively, if the existing ServiceMix service/endpoint represents a ServiceMix binding component, you could replace the ServiceMix binding component with an equivalent Mediation Router component (thus bypassing the NMR).

---

## The ServiceMix Camel module

The integration between Mediation Router and ServiceMix is provided by the *ServiceMix Camel* module. This module is provided with ServiceMix, but actually implements a plug-in for the Mediation Router product. From the perspective of Mediation Router, the ServiceMix Camel module provides the *JB1 component* (see [JB1](#) in the *Component Reference* and [JB1 Component](#) [<http://activemq.apache.org/camel/jbi.html>]). When the ServiceMix Camel module is included on your CLASSPATH, you can access the JBI component by defining Mediation Router endpoint URIs with the `jbi:` component prefix.

---

## Translating ServiceMix URIs into Mediation Router endpoint URIs

ServiceMix defines a flexible format for defining URIs, which is described in detail in [ServiceMix URIs](#) [<http://servicemix.apache.org/uris.html>]. To translate a ServiceMix URI into a Mediation Router endpoint URI, simply prefix it with `jbi:.`. In other words, the general format for Mediation Router URIs that access a ServiceMix URI, *ServiceMixURI*, through the JBI component is as follows:

```
jbi:ServiceMixURI
```

For example, consider the following configuration of the [static recipient list](#) pattern in ServiceMix EIP. The `eip:exchange-target` elements define some targets using the ServiceMix URI format.

```
<beans xmlns:sm="http://servicemix.apache.org/config/1.0"
  xmlns:eip="http://servicemix.apache.org/eip/1.0"
  xmlns:test="http://iona.com/demos/test" >
  ...
  <eip:static-recipient-list service="test:recipients" end
point="endpoint">
    <eip:recipients>
      <eip:exchange-target uri="service:test:messageFilter"
/>
      <eip:exchange-target uri="service:test:trace4" />
    </eip:recipients>
  </eip:static-recipient-list>
  ...
</beans>
```

When the preceding ServiceMix configuration is mapped to an equivalent Mediation Router configuration, you get the following route:

```
<route>
  <from uri="jbi:endpoint:test:recipients:endpoint"/>
  <to uri="jbi:service:test:messageFilter"/>
  <to uri="jbi:service:test:trace4"/>
</route>
```

Where the target endpoint URIs in this route are derived from the corresponding ServiceMix URIs by adding the `jbi:` prefix at the start.

## Representing ServiceMix targets as Mediation Router endpoint URIs

ServiceMix URIs are not the only format for specifying ServiceMix targets. For example, the source of messages for a static recipient list pattern in ServiceMix can be specified using a combination of `service` and `endpoint` attributes, as follows:

```
<eip:static-recipient-list service="test:recipients" end
point="endpoint">
```

In order to translate this ServiceMix target into a Mediation Router endpoint URI, start by reformatting it as a ServiceMix URI:

```
endpoint:test:recipients:endpoint
```

Then add the `jbi:` prefix to turn it into a Mediation Router endpoint URI, as follows:

```
jbi:endpoint:test:recipients:endpoint
```

For full details of how to reformat ServiceMix targets as ServiceMix URIs, see [ServiceMix URIs](http://servicemix.apache.org/uris.html) [http://servicemix.apache.org/uris.html].

---

## Replacing ServiceMix bindings with Mediation Router components

Instead of using the Mediation Router JBI component to route all your messages through the ServiceMix NMR, you could use one of the many supported Mediation Router components to connect directly to a source or target endpoint. In particular, when sending messages to an external endpoint, it is frequently more efficient to send the messages directly through a Mediation Router component rather than sending them through the NMR and a ServiceMix binding.

For details of all the Mediation Router components that are available, see [Components](#) in the *Component Reference* and [Mediation Router Components](http://activemq.apache.org/camel/components.html) [http://activemq.apache.org/camel/components.html].

DRAFT

## Common Elements

---

### Overview

When configuring ServiceMix EIP patterns in a ServiceMix configuration file, there are some common elements that recur in many of the pattern schemas. This section provides a brief overview of these common elements and explains how they can be mapped to equivalent constructs in Mediation Router.

---

### Exchange target

All of the patterns supported by ServiceMix EIP use the `eip:exchange-target` element to specify JBI target endpoints.

[Table A.1 on page 115](#) shows some examples of how to map some sample `eip:exchange-target` elements to Mediation Router endpoint URIs.

**Table A.1. Mapping the Exchange Target Element**

ServiceMix EIP Target	Mediation Router Endpoint URI
<code>&lt;eip:exchange-target interface="HelloWorld" /&gt;</code>	<code>jbi:interface:HelloWorld</code>
<code>&lt;eip:exchange-target service="test:HelloWorldService" /&gt;</code>	<code>jbi:service:test:HelloWorldServ</code>
<code>&lt;eip:exchange-target service="test:HelloWorldService" endpoint="secure" /&gt;</code>	<code>jbi:service:test:HelloWorldServ</code>
<code>&lt;eip:exchange-target uri="service:test:HelloWorldService" /&gt;</code>	<code>jbi:service:test:HelloWorldServ</code>

---

### Predicates

The ServiceMix EIP component lets you define predicate expressions in the XPath language (for example, XPath predicates can appear in `eip:xpath-predicate` elements or in `eip:xpath-splitter` elements, where the XPath predicate is specified using an `xpath` attribute).

ServiceMix XPath predicates can easily be migrated to equivalent constructs in Mediation Router: that is, either the `xpath` element (in XML configuration)

or the `xpath()` command (in Java DSL). For example, the message filter pattern in Mediation Router can incorporate an XPath predicate as follows:

```
<route>
  <from uri="jbi:endpoint:test:messageFilter:endpoint">
  <filter>
    <xpath>count (/test:world) = 1</xpath>
    <to uri="jbi:service:test:trace3"/>
  </filter>
</route>
```

Where the `xpath` element specifies that only messages containing the `test:world` element will pass through the filter.



## Note

Mediation Router also supports a wide range of other scripting languages (such as XQuery, PHP, Python, Ruby, and so on), which can be used to define predicates. For details of all the supported predicate languages, see [Languages for Expressions and Predicates](#) in the *Defining Routes* and [Languages for Expressions and Predicates](#) in the *Defining Routes* .

## Namespace contexts

When using XPath predicates in the ServiceMix EIP configuration, it is necessary to define a namespace context using the `eip:namespace-context` element. The namespace can then be referenced using a `namespaceContext` attribute.

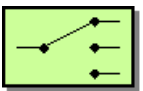
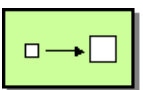




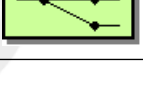
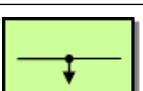
When ServiceMix EIP configuration is migrated to Mediation Router, however, there is no need to define namespace contexts, because Mediation Router allows you to define XPath predicates without referencing a namespace context. Hence, you can simply drop the `eip:namespace-context` elements when you migrate to Mediation Router.

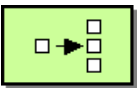
## ServiceMix EIP Patterns

### Supported patterns

The patterns supported by ServiceMix EIP are shown in [Table A.2 on page 117](#).

**Table A.2. ServiceMix EIP Patterns**

	<a href="#">Content-Based Router</a>	How do we handle a situation where the implementation of a single logical function (e.g., inventory check) is spread across multiple physical systems?
	<a href="#">Content Enricher</a>	How do we communicate with another system if the message originator does not have all the required data items available?
	<a href="#">Message Filter</a>	How can a component avoid receiving uninteresting messages?
	<a href="#">Pipeline</a>	How can we perform complex processing on a message while maintaining independence and flexibility?
	<a href="#">Resequencer</a>	How can we get a stream of related but out-of-sequence messages back into the correct order?
	<a href="#">Split Aggregator</a>	How do we combine the results of individual, but related messages so that they can be processed as a whole?
	<a href="#">Static Recipient List</a>	How do we route a message to a list of specified recipients?
	<a href="#">Static Routing Slip</a>	How do we route a message consecutively through a series of processing steps?
	<a href="#">Wire Tap</a>	How do you inspect messages that travel on a point-to-point channel?

	<p><a href="#">XPath Splitter</a></p>	<p>How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?</p>
---	---------------------------------------	--

DRAFT

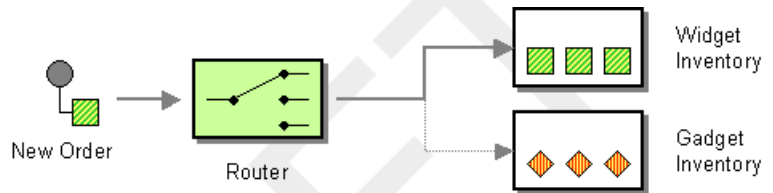


# Content-Based Router

## Overview

A *content-based router* enables you to route messages to the appropriate destination, where the routing decision is based on the message contents. This pattern maps to the corresponding [content-based router on page 52](#) pattern in FUSE Mediation Router.

**Figure A.1. Content-Based Router Pattern**



## Example ServiceMix EIP route

The following example shows how to define a content-based router using the ServiceMix EIP component. Incoming messages are routed to the `http://test/pipeline/endpoint` endpoint, if a `test:echo` element is present in the message body, and to the `test:recipients` endpoint, otherwise:

```
<eip:content-based-router service="test:router" endpoint="en
dpoint">
  <eip:rules>
    <eip:routing-rule>
      <eip:predicate>
        <eip:xpath-predicate xpath="count(/test:echo) = 1"
namespaceContext="#nsContext" />
      </eip:predicate>
      <eip:target>
        <eip:exchange-target uri="endpoint:ht
tp://test/pipeline/endpoint" />
      </eip:target>
    </eip:routing-rule>
    <eip:routing-rule>
      <!-- There is no predicate, so this is the default des
tination -->
      <eip:target>
        <eip:exchange-target service="test:recipients" />
      </eip:target>
    </eip:routing-rule>
  </eip:rules>
</eip:content-based-router>
```

```
</eip:rules>
</eip:content-based-router>
```

### Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>
  <from uri="jbi:endpoint:test:router:endpoint"/>
  <choice>
    <when>
      <xpath>count(/test:echo) = 1</xpath>
      <to uri="jbi:endpoint:http://test/pipeline/endpoint"/>
    </when>
    <otherwise>
      <!-- This is the default destination -->
      <to uri="jbi:service:test:recipients"/>
    </otherwise>
  </choice>
</route>
```

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

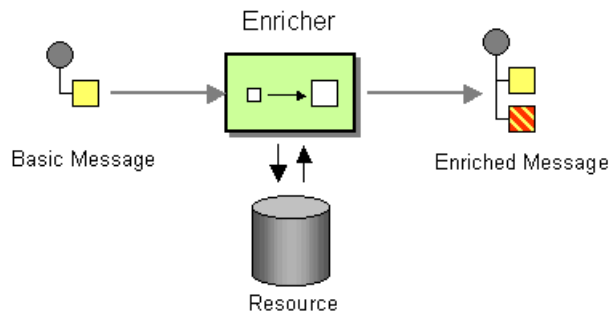
```
from("jbi:endpoint:test:router:endpoint").
  choice().when(xpath("count(/test:echo) = 1")).to("jbi:en
dpoint:http://test/pipeline/endpoint").
  otherwise().to("jbi:service:test:recipients");
```

# Content Enricher

## Overview

A *content enricher* is a pattern for augmenting a message with missing information. The ServiceMix EIP content enricher is more or less equivalent to a pipeline that adds missing data as the message passes through an enricher target. Consequently, when migrating to Mediation Router, you can re-implement the ServiceMix content enricher as a Mediation Router pipeline.

**Figure A.2. Content Enricher Pattern**



## Example ServiceMix EIP route

The following example shows how to define a content enricher using the ServiceMix EIP component. Incoming messages pass through the enricher target, `test:additionalInformationExtractor`, which adds some missing data to the message before the message is sent on to its ultimate destination, `test:myTarget`.

```
<eip:content-enricher service="test:contentEnricher" endpoint="endpoint">
  <eip:enricherTarget>
    <eip:exchange-target service="test:additionalInformationExtractor" />
  </eip:enricherTarget>
  <eip:target>
    <eip:exchange-target service="test:myTarget" />
  </eip:target>
</eip:content-enricher>
```

## Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>
  <from uri="jbi:endpoint:test:contentEnricher:endpoint"/>
  <to uri="jbi:service:test:additionalInformationExtractor"/>

  <to uri="jbi:service:test:myTarget"/>
</route>
```

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

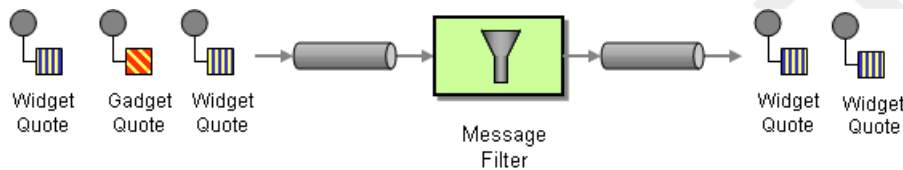
```
from("jbi:endpoint:test:contentEnricher:endpoint").
  to("jbi:service:test:additionalInformationExtractor").
  to("jbi:service:test:myTarget");
```

# Message Filter

## Overview

A *message filter* is a processor that eliminates undesired messages based on specific criteria. Filtering is controlled by specifying a predicate in the filter: when the predicate is `true`, the incoming message is allowed to pass; otherwise, it is blocked. This pattern maps to the corresponding [message filter on page 54](#) pattern in FUSE Mediation Router.

**Figure A.3. Message Filter Pattern**



## Example ServiceMix EIP route

The following example shows how to define a message filter using the ServiceMix EIP component. Incoming messages are passed through a filter mechanism that blocks messages that lack a `test:world` element.

```
<eip:message-filter service="test:messageFilter" endpoint="en
dpoint">
  <eip:target>
    <eip:exchange-target service="test:trace3" />
  </eip:target>
  <eip:filter>
    <eip:xpath-predicate xpath="count(/test:world) = 1"
namespaceContext="#nsContext"/>
  </eip:filter>
</eip:message-filter>
```

## Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>
  <from uri="jbi:endpoint:test:messageFilter:endpoint">
    <filter>
      <xpath>count(/test:world) = 1</xpath>
    <to uri="jbi:service:test:trace3"/>
  </from>
</route>
```

```
</filter>  
</route>
```

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

```
from("jbi:endpoint:test:messageFilter:endpoint").  
  filter(xpath("count(/test:world) = 1")).  
  to("jbi:service:test:trace3");
```

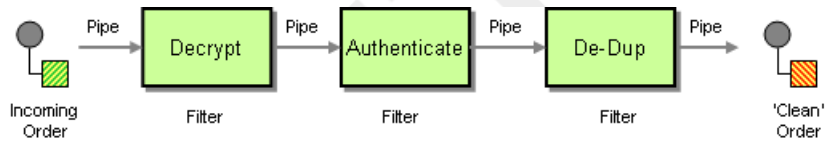
DRAFT

# Pipeline

## Overview

The ServiceMix EIP *pipeline* pattern is used to pass messages through a single transformer endpoint, where the transformer's input is taken from the source endpoint and the transformer's output is routed to the target endpoint. This pattern is thus a special case of the more general FUSE Mediation Router [pipes and filters on page 27](#) pattern, which enables you to pass an *In* message through *multiple* transformer endpoints.

**Figure A.4. Pipes and Filters Pattern**



## Example ServiceMix EIP route

The following example shows how to define a pipeline using the ServiceMix EIP component. Incoming messages are passed into the transformer endpoint, `test:decrypt`, and the output from the transformer endpoint is then passed into the target endpoint, `test:plaintextOrder`.

```
<eip:pipeline service="test:pipeline" endpoint="endpoint">
  <eip:transformer>
    <eip:exchange-target service="test:decrypt" />
  </eip:transformer>
  <eip:target>
    <eip:exchange-target service="test:plaintextOrder" />
  </eip:target>
</eip:pipeline>
```

## Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>
  <from uri="jbi:endpoint:test:pipeline:endpoint"/>
  <to uri="jbi:service:test:decrypt"/>
</route>
```

```
<to uri="jbi:service:test:plaintextOrder"/>
</route>
```

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

```
from("jbi:endpoint:test:pipeline:endpoint").
    pipeline("jbi:service:test:decrypt", "jbi:service:test:plaintextOrder");
```

DRAFT

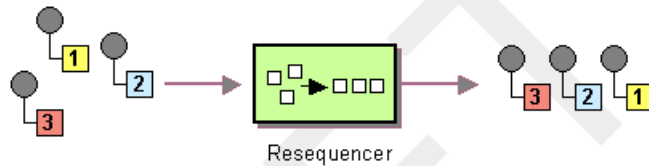


# Resequencer

## Overview

The *resequencer* pattern enables you to resequence messages according to the sequence number stored in an NMR property. The ServiceMix EIP resequencer pattern maps to the Mediation Router [resequencer](#) on page 66 configured with the *stream resequencing* algorithm.

**Figure A.5. Resequencer Pattern**



## Sequence number property

The sequence of messages emitted from the resequencer is determined by the value of the sequence number property: messages with a low sequence number are emitted first and messages with a higher number are emitted later. By default, the sequence number is read from the `org.apache.servicemix.eip.sequence.number` property in ServiceMix. But you can customize the name of this property using the `eip:default-comparator` element in ServiceMix.

The equivalent concept in Mediation Router is a *sequencing expression*, which can be any message-dependent expression. When migrating from ServiceMix EIP, you would normally define an expression that extracts the sequence number from a header (a Mediation Router header is equivalent to an NMR message property). For example, to extract a sequence number from a `seqnum` header, you could use the simple expression, `header.seqnum`.

## Example ServiceMix EIP route

The following example shows how to define a resequencer using the ServiceMix EIP component.

```
<eip:resequencer
  service="sample:Resequencer"
  endpoint="ResequencerEndpoint"
  comparator="#comparator"
  capacity="100"
  timeout="2000">
  <eip:target>
```

```

    <eip:exchange-target service="sample:SampleTarget" />
  </eip:target>
</eip:resequencer>

<!-- Configure default comparator with custom sequence number
property -->
<eip:default-comparator id="comparator" sequenceNumberKey="seqnum"/>

```

---

### Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```

<route>
  <from uri="jbi:endpoint:sample:Resequencer:ResequencerEndpoint"/>
  <resequencer>
    <simple>header.seqnum</simple>
    <to uri="jbi:service:sample:SampleTarget" />
    <stream-config capacity="100" timeout="2000"/>
  </resequencer>
</route>

```

---

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

```

from("jbi:endpoint:sample:Resequencer:ResequencerEndpoint").
  resequencer(header("seqnum")).
  stream(new StreamResequencerConfig(100, 2000L)).
  to("jbi:service:sample:SampleTarget");

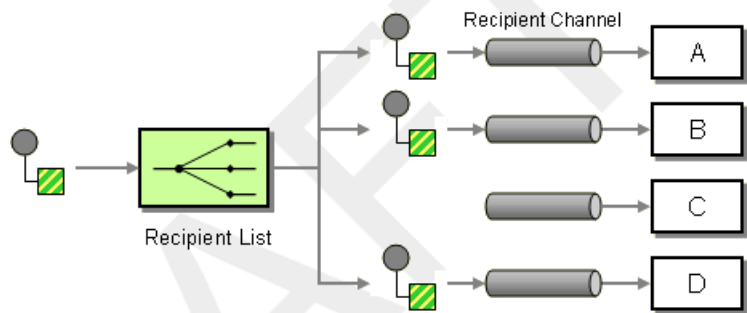
```

## Static Recipient List

### Overview

A *recipient list* is a type of router that sends each incoming message to multiple different destinations. The ServiceMix EIP recipient list is restricted to processing *InOnly* and *RobustInOnly* exchange patterns. Moreover, the list of recipients must be static. This pattern maps to the [recipient list on page 56](#) with fixed destination pattern in FUSE Mediation Router.

**Figure A.6. Static Recipient List Pattern**



### Example ServiceMix EIP route

The following example shows how to define a static recipient list using the ServiceMix EIP component. Incoming messages are copied to the `test:messageFilter` endpoint and to the `test:trace4` endpoint.

```
<eip:static-recipient-list service="test:recipients" endpoint="endpoint">
  <eip:recipients>
    <eip:exchange-target service="test:messageFilter" />
    <eip:exchange-target service="test:trace4" />
  </eip:recipients>
</eip:static-recipient-list>
```

### Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>
  <from uri="jbi:endpoint:test:recipients:endpoint"/>
  <to uri="jbi:service:test:messageFilter"/>
  <to uri="jbi:service:test:trace4"/>
</route>
```



## Note

The preceding route configuration appears to have the same syntax as a Mediation Router pipeline pattern. The crucial difference is that the preceding route is intended for processing *InOnly* message exchanges, which are processed in a slightly different way—see [Pipes and Filters on page 27](#) for more details.

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

```
from("jbi:endpoint:test:recipients:endpoint").  
    to("jbi:service:test:messageFilter", "jbi:service:test:trace4");
```

## Static Routing Slip

---

### Overview

The *static routing slip* pattern in the ServiceMix EIP component is used to route an *InOut* message exchange through a series of endpoints. Semantically, it is equivalent to the [pipeline on page 27](#) pattern in FUSE Mediation Router.

---

### Example ServiceMix EIP route

The following example shows how to define a static routing slip using the ServiceMix EIP component. Incoming messages pass through each of the endpoints, `test:procA`, `test:procB`, and `test:procC`, where the output of each endpoint is connected to the input of the next endpoint in the chain. The final endpoint, `tets:procC`, sends its output (*Out* message) back to the caller.

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

---

### Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>
  <from uri="jbi:endpoint:test:routingSlip:endpoint"/>
  <to uri="jbi:service:test:procA"/>
  <to uri="jbi:service:test:procB"/>
  <to uri="jbi:service:test:procC"/>
</route>
```

---

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

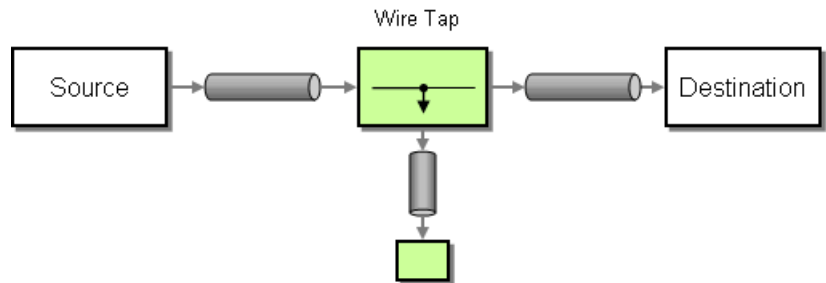
```
from("jbi:endpoint:test:routingSlip:endpoint")
  .pipeline("jbi:service:test:procA", "jbi:service:test:procB", "jbi:service:test:procC");
```

# Wire Tap

## Overview

The *wire tap* pattern allows you to route messages to a separate tap location before it is forwarded to the ultimate destination. The ServiceMix EIP wire tap pattern maps to the [wire tap on page 110](#) pattern in Mediation Router.

**Figure A.7. Wire Tap Pattern**



## Example ServiceMix EIP route

The following example shows how to define a wire tap using the ServiceMix EIP component. The *In* message from the source endpoint is copied to the *In*-listener endpoint, before being forwarded on to the target endpoint. If you want to monitor any returned *Out* messages or *Fault* messages from the target endpoint, you would also need to define an *Out* listener (using the `eip:outListener` element) and a *Fault* listener (using the `eip:faultListener` element).

```

<eip:wire-tap service="test:wireTap" endpoint="endpoint">
  <eip:target>
    <eip:exchange-target service="test:target" />
  </eip:target>
  <eip:inListener>
    <eip:exchange-target service="test:trace1" />
  </eip:inListener>
</eip:wire-tap>

```

## Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```

<route>
  <from uri="jbi:endpoint:test:wireTap:endpoint"/>

```

```
<to uri="jbi:service:test:trace1"/>  
<to uri="jbi:service:test:target"/>  
</route>
```

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

```
from("jbi:endpoint:test:wireTap:endpoint").to("jbi:service:test:trace1", "jbi:service:test:target");
```

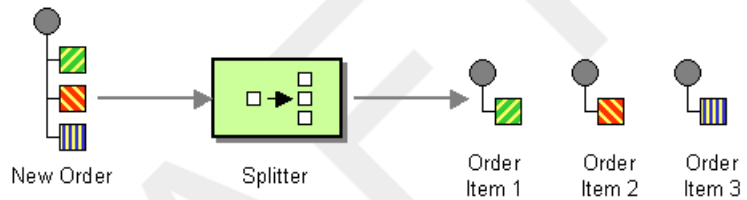
DRAFT

# XPath Splitter

## Overview

A *splitter* is a type of router that splits an incoming message into a series of outgoing messages, where each of the messages contains a piece of the original message. The ServiceMix EIP XPath splitter pattern is restricted to using the *InOnly* and *RobustInOnly* exchange patterns. The expression that defines how to split up the original message is defined in the XPath language. The XPath splitter pattern maps to the [splitter on page 59](#) pattern in Mediation Router.

**Figure A.8. XPath Splitter Pattern**



## Forwarding NMR attachments and properties

The `eip:xpath-splitter` element supports a `forwardAttachments` attribute and a `forwardProperties` attribute, either of which can be set to `true`, if you want the splitter to copy the incoming message's attachments or properties to the outgoing messages. The corresponding splitter pattern in Mediation Router does not support any such attributes. By default, the incoming message's headers are copied to each of the outgoing messages by the Mediation Router splitter.

## Example ServiceMix EIP route

The following example shows how to define a splitter using the ServiceMix EIP component. The specified XPath expression, `/*/*`, would cause an incoming message to split at every occurrence of a nested XML element (for example, the `/foo/bar` and `/foo/car` elements would be split into distinct messages).

```
<eip:xpath-splitter service="test:xpathSplitter" endpoint="endpoint"
    xpath="/*/*" namespaceContext="#nsContext">
  <eip:target>
    <eip:exchange-target uri="service:http://test/router" />
  </eip:target>
</eip:xpath-splitter>
```



```
</eip:target>  
</eip:xpath-splitter>
```

---

### Equivalent Mediation Router XML route

The following example shows how to define an equivalent route using Mediation Router XML configuration:

```
<route>  
  <from uri="jbi:endpoint:test:xpathSplitter:endpoint"/>  
  <splitter>  
    <xpath>/*/*</xpath>  
    <to uri="jbi:service:http://test/router"/>  
  </splitter>  
</route>
```

---

### Equivalent Mediation Router Java DSL route

The following example shows how to define an equivalent route using the Mediation Router Java DSL:

```
from("jbi:endpoint:test:xpathSplitter:endpoint").  
  splitter(xpath("/*/*")).to("jbi:service:ht  
tp://test/router");
```

DRAFT