## Table of Contents


1. Enterprise Integration Patterns (EIPs) Lab
   1.1. Route the Content of CSV/XML Files Using the CBR Pattern
   1.2. Aggregate the Content of XML Individual Payments
   1.3. Distribute the Exchange to a List of Recipients Calculated Based on a Header
   1.4. Enrich a Message with Another Camel Route


# 1. Enterprise Integration Patterns (EIPs) Lab

**Goals**

- Design Apache Camel routes using some of the most common Enterprise Integration Patterns (EIPs)
- Understand how you can aggregate messages after being split
- Distribute data, using content based routing, to a list of recipients
- Enrich a message with an Apache Camel route

**Lab Assets**

The lab exercises and solutions are available in the following zip archives:

- https://github.com/gpe-mw-training/camel-labs/archive/v0.3-exercise.zip.
- https://github.com/gpe-mw-training/camel-labs/archive/v0.3-solution.zip.

## 1.1. Route the Content of CSV/XML Files Using the CBR Pattern

This exercise uses the Content Based Router pattern to route the content of a file to a different destination according to the file content or file name. The lab_assets.zip archive contains the code for the two Apache Camel routes as well as the `camel-context.xml` file (containing bean definitions).

In this lab exercise, you complete the following:

- Design the routes
- Define and inject the endpoints within the bean
- Introduce the content based router using either XPath or Simple as the expression language

The code for the two Apache Camel routes is as follows:

```
from(SOURCE_OF_THE_XML_FILES)
  .choiceBasedOnXpathExpression // /pay:Payments/pay:Currency = 'EUR'
    .toFile("EUR_TARGET_DIRECTORY")
    .ORtoFile("USD_TARGET_DIRECTORY") // /pay:Payments/pay:Currency = 'USD'
    .otherWiseToFile("OTHER_TARGET_DIRECTORY")

from(SOURCE_OF_THE_CSVFILES)
  .choiceBasedOnSimpleExpression // ${file:onlyname} == 'EUPayments.txt'
    .toFile("EUR_TARGET_DIRECTORY")
    .ORtoFile("USD_TARGET_DIRECTORY") // ${file:onlyname} == 'USPayments.txt'
    .otherWiseToFile("OTHER_TARGET_DIRECTORY")
```

Follow these steps:

1. Create the project:
   a. Import the lab assets (in `lab_assets.zip`) into a new project (that you will create) titled `camel-cbr`.
   b. In JBoss Developer Studio, use **Project Explorer** to open the `camel-cbr` project.
   c. Review the project contents:
      - XML files
      - `data/in/csv`
      - `data/in/xml`
      - Two `RouteBuilder` classes
2. Define the endpoints:
   a. Edit the file `src/main/META-INF/spring/camel-context.xml` by specifying which `package` to use.
   b. Define the endpoint beans within the `<camelContext/>` tag.
   c. Specify these directory locations: `src/main/data/in/csv` and `src/main/data/in/xml`, where files are located and are polled for.
   d. Using the source directory URI defined in the `cbr.properties` file, assign it to the respective endpoint as follows:

`<endpoint id="sourceDirectoryXml" uri="{{sourceDirectoryXmlUri}}"/>`.

e. Do the same for the destination directory URI endpoints, where you will publish the files based on their content. Again, refer to the `cbr.properties` file for the URI.

Each endpoint is referenced by its ID during Exchange routing and delivery.

3. Inject the endpoints:

a. Open the class `src/main/java/com/redhat/gpe/training/camel/SimpleCBRRoute.java`.

b. Add the `@EndpointInject` annotation to allow lookups of the four endpoints that this route uses (see their definitions in the `camel-context.xml` file).

4. Design the route based on the code snippet shown at the beginning of this lab (where the CBR uses the name of the file to take a decision) as follows:

a. Repeat the previous steps to create a route, and name the route `XPathCBRRoute`.

Remember to register the namespace for the XPath expression of the CBR.

b. Enable logging by adding the log processor.

5. Check your work:

a. Launch the project using `mvn camel:run`.

b. Verify that both the CBR and XML files were processed, and that new output files were generated under `src/main/data/out/csv` or `src/main/data/out/xml`.

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route1 started and consuming from: Endpoint[file://src/main/data/in/csv?noop=true]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route2 started and consuming from: Endpoint[file://src/main/data/in/xml?noop=true]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 2 routes, of which 2 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.297 seconds
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [route2] - >> Processing XML files - EUPayments.xml <<
Camel (camel-1) thread #0 - file://src/main/data/in/csv INFO [route1] - >> Processing CSV files - EUPayments.txt <<
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [org.apache.camel.builder.xml.XPathBuilder] - Created default XPathFactory com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl@37773c44
Camel (camel-1) thread #0 - file://src/main/data/in/csv INFO [route1] - This is an Euro Payment: EUPayments.txt
Camel (camel-1) thread #0 - file://src/main/data/in/csv INFO [route1] - >> Processing CSV files - USPayments.txt <<
Camel (camel-1) thread #0 - file://src/main/data/in/csv INFO [route1] - This is an USD Payment: USPayments.txt
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [route2] - This is an Euro Payment: EUPayments.xml
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [route2] - >> Processing XML files - UnknownPayments.xml <<
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [route2] - This is an Other Currency Payment: UnknownPayments.xml
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [route2] - >> Processing XML files - USPayments.xml <<
Camel (camel-1) thread #1 - file://src/main/data/in/xml INFO [route2] - This is an USD Payment: USPayments.xml
```

**Figure 1. Console output for CBR**

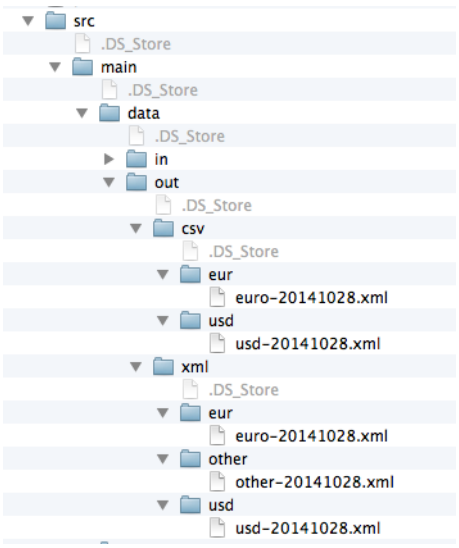


**Figure 2. Folders**

## 1.2. Aggregate the Content of XML Individual Payments

This exercise combines two EIPs (`Splitter` and `Aggregator`) in a payment scenario. You use them to split the XML content of a file that contains `<Payment/>` tags, each tag representing a payment record. All records are aggregated using an XPath expression, and the aggregate contains a new collection of `<Payment>` entries, where the respective payee (defined within the `</to>` tag) remains the same.

The code for the Camel Route appears below and is also included in lab_assets.zip.

You now complete these steps:

- Design the route to both split and aggregate the content
- Implement the aggregation strategy in a Java class

The code for the Apache Camel Route is as follows:

```
from(SOURCE_OF_THE_XML_FILES)
  .split().xpath // "/p:Payments/p:Payment"
    .log("BODY PROCESSED")
    .convertToString()
    .aggregate()
       .aggregationStrategy  // 1
       .xpath() // 2 - "/p:Payment/p:to"
       .completionTimeOut // 3
    .toFile("OUT_TARGET_DIRECTORY")
```

Follow these steps:

1. Open the project:
   a. In JBoss Developer Studio, use **Project Explorer** to open the `camel-aggregator` project.
   b. Review the following artifacts:
      - `camel-context.xml`
      - Files under `data/in/xml`
      - `BodyAppenderAggregator` class
      - `PaymentsAggregatorRoute` class
2. Design the route:
   a. Edit the `PaymentsAggregatorRoute.java` file to design the Apache Camel route.
   b. Use the `@EndpointInject` approach to define `src/main/data/in/xml` as the source directory within the `<from/>` tag.
   c. Split the file content into individual payments using the XPath expression language.
   d. Register the namespace required to handle the XML payment data as follows:
   `Namespaces ns = new Namespaces("p", "http://training.gpe.redhat.com/payment").add("xsd", "http://ww`

   

   The `BodyAppenderAggregator` class is responsible for data aggregation and that you use the XPath expression to compare tags in the file content.

   e. Log the content of the body (which has been split) as a console message.
   f. Aggregate the individual `<Payment/>` tags that are received into a new aggregate using `:` as a delimiter.
   g. Set the `timeout` (waiting for completion) value to `5000ms`.
   h. Send the aggregated content to the destination `file` endpoint.
   i. Design the route that aggregates the old Exchange with the new Exchange received. The old and new exchanges are joined to form a new string.
3. Check your work:
   a. Launch the project using `mvn camel:run`.
   b. Check the log for the status of the data aggregation.

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route1 started and consuming from: Endpoint[file://src/main/data/in?noop=true]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 1 routes, of which 1 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.304 seconds
Camel (camel-1) thread #1 - file://src/main/data/in INFO [org.apache.camel.builder.xml.XPathBuilder] - Created default XPathFactory com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl@6decbf94
Camel (camel-1) thread #1 - file://src/main/data/in INFO [route1] -
Got separated payment with this content:
<tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>charles-EUR</tns:from>
        <tns:to>jeff</tns:to>
        <tns:amount>1000000.0</tns:amount>
    </tns:Payment>
which is now being handed over to the aggregator

Camel (camel-1) thread #1 - file://src/main/data/in INFO [com.redhat.gpe.training.camel.BodyAppenderAggregator] - Old: null, new: <tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>charles-EUR</tns:from>
        <tns:to>jeff</tns:to>
        <tns:amount>1000000.0</tns:amount>
    </tns:Payment>
Camel (camel-1) thread #1 - file://src/main/data/in INFO [route1] -
Got separated payment with this content:
<tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>jeff-EUR</tns:from>
        <tns:to>chad</tns:to>
        <tns:amount>20.0</tns:amount>
    </tns:Payment>
which is now being handed over to the aggregator
```

**Figure 3. Data aggregation status**

   c. Check the two new files created in the `src/main/data/out` directory.

```
Got aggregated payments with this content:
<tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>charles-EUR</tns:from>
        <tns:to>jeff</tns:to>
        <tns:amount>1000000.0</tns:amount>
    </tns:Payment><tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>hong-EUR</tns:from>
        <tns:to>jeff</tns:to>
        <tns:amount>12345.0</tns:amount>
    </tns:Payment><tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>chad-USD</tns:from>
        <tns:to>jeff</tns:to>
        <tns:amount>1000000.0</tns:amount>
    </tns:Payment><tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>jingjing-USD</tns:from>
        <tns:to>jeff</tns:to>
        <tns:amount>10.0</tns:amount>
    </tns:Payment>
```

**Figure 4. First file created**

```
Got aggregated payments with this content:
<tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>jeff-EUR</tns:from>
        <tns:to>chad</tns:to>
        <tns:amount>20.0</tns:amount>
    </tns:Payment><tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>bernard-EUR</tns:from>
        <tns:to>chad</tns:to>
        <tns:amount>42.0</tns:amount>
    </tns:Payment><tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>rachel-USD</tns:from>
        <tns:to>chad</tns:to>
        <tns:amount>78.0</tns:amount>
    </tns:Payment><tns:Payment xmlns:tns="http://training.gpe.redhat.com/payment">
        <tns:from>rachel-USD</tns:from>
        <tns:to>chad</tns:to>
        <tns:amount>13.0</tns:amount>
    </tns:Payment>
```

**Figure 5. Second file created**

## 1.3. Distribute the Exchange to a List of Recipients Calculated Based on a Header

In this exercise, you use the `recipientlist` EIP to send an Apache Camel Exchange to two `direct` endpoints using a header string called `recipient` that specifies the list of recipients and how the processor should handle the list when it is received. The code provided in the project includes the `RouteBuilder` class and the Spring `camel-context.xml` file.

You now complete these steps:

- Design the route and the recipient strategy
- Add the routes that contain the `direct` endpoints that the recipient processor invokes

The code for the Apache Camel routes is as follows:

```
from(TIMER_PERIOD_EVERY_5s)
  .setABody_using_constant_language_and_say_hello
  .set_A_Header_where_name_is_recipients_and_value("direct:a;direct:b")
  .call_the_recipient_list_USING_recipients_header_and_sprcify_the _token_to_be_used_to_split(";")

from(DIRECT_A)
  .log(>> Recipient A called - ${body})

from(DIRECT_B)
  .log(>> Recipient A called - ${body})
```

Follow these steps:

1. Open the project:
   a. In JBoss Developer Studio, use **Project Explorer** to open the `camel-recipientlist` project.
   b. Review the contents of the project:
      - Spring Camel XML file
      - `RouteBuilder` class.
2. Add the first class by editing the `RecipientRoute` class as follows:
   a. Add the `from("")` endpoint definition using the `timer` component. Remember to pass the period of `5 seconds` as a parameter.
   b. Add the `.setBody` processor that sends the message `Say Hello for recipient` using the `constant` language.

c. Add a `Header` named `recipients`, which contains the value `direct:a;direct:b`.

d. Invoke the `recipientList()` processor and pass, using the header language, the header `recipients` as a parameter.

e. Specify the use of `tokenize` in splitting the header recipients.

3. Add the two routes that consume data from a `direct` endpoint:

a. Specify the `direct:a` endpoint as the source and log a message `>> Recipient A called - ${body}` when the endpoint is invoked.

b. Repeat this process with `direct:b` as the source.

4. Check your work:

a. Launch the project using `mvn camel:run`.

b. Use the console log to validate the results:

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route3 started and consuming from: Endpoint[direct://b]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 3 routes, of which 3 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: mycontext) started in 0.365 seconds
Camel (mycontext) thread #0 - timer://recipient INFO [route2] - >> Recipient A called - Say Hello for recipient
Camel (mycontext) thread #0 - timer://recipient INFO [route3] - >> Recipient B called - Say Hello for recipient
Camel (mycontext) thread #0 - timer://recipient INFO [route2] - >> Recipient A called - Say Hello for recipient
Camel (mycontext) thread #0 - timer://recipient INFO [route3] - >> Recipient B called - Say Hello for recipient
```

**Figure 6. Console results for the recipient list**

## 1.4. Enrich a Message with Another Camel Route

During this exercise, you use the `enrich` EIP to enrich a message with the output from an Apache Camel route. The `camel-recipientlist` project contains the `RouteBuilder` class definition and the Spring `camel-context.xml` file.

In this lab exercise, you design the two Camel routes:

- A route that sends a message to the `enrich` processor, upon a timer triggering
- A route that sends the response message when invoked by the `enrich` processor.

The code for the Apache Camel routes is as follows:

```
from(TIMER_PERIOD_EVERY_5s)
  .setABody_using_constant_language_and_add_("message")
  .log_to_say_(">> Before enrichment. My body is : ${body}")
  .call_the_enricher_using_direct_endpoint_AND_pass_as_parameter_the_aggregation_strategy_used_to_concatenate_the_content
  .log_to_say(">> After enrichment. My body is : ${in.body}")

from("DIRECT_RESOURCE")
     .setExchangePatterntoINOUT
     .transform().constant("blah");
```

Follow these steps:

1. Open the project:

a. In JBoss Developer Studio, use **Project Explorer** to open the `camel-enricher` project.

b. Review the project contents:

- Spring Camel XML file
- `SampleAggregator` class
- `EnricherRoute` class

2. Add the first class by editing the `EnricherRoute` class as follows:

a. Add the `from("")` endpoint definition using the `timer` component. Remember to pass the period of `5 seconds` as a parameter.

b. Add the `.setBody` processor to set the message body using the `constant` language.

c. Invoke the `enrich()` processor and pass the `aggregation strategy` as a parameter. Use `direct:resource` as the endpoint.

d. Add the last processor to the end of the route to log this response to the console: `>> After enrichment. My body is : ${in.body}`.

3. Add the second route that consumes data from the `direct:resource` endpoint:

a. Add a `direct:resource` endpoint as the source.

b. Set the `exchangePattern` to `InOut`.

c. Add the `transform` processor, using the `constant` language, to set the response to `blah`.

**Check your work:**

a. Launch the project using `mvn camel:run`.

b. Analyze the log within the console and validate the result.

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route1 started and consuming from: Endpoint[timer://enrich?period=5s]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route2 started and consuming from: Endpoint[direct://resource]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 2 routes, of which 2 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.328 seconds
Camel (camel-1) thread #0 - timer://enrich INFO [route1] - >> Before enrichment. My body is : message
Camel (camel-1) thread #0 - timer://enrich INFO [route1] - >> After enrichment. My body is : message:blah
Camel (camel-1) thread #0 - timer://enrich INFO [route1] - >> Before enrichment. My body is : message
Camel (camel-1) thread #0 - timer://enrich INFO [route1] - >> After enrichment. My body is : message:blah
```

**Figure 7. Console results after enrichment**

Last updated 2015-11-12 12:04:12 EST