Table of Contents

1. Components Lab

Goals

- Use the Apache CXF Framework to create an Apache Camel Route that exposes a Web Service.
- Use the Apache Camel SQL component to poll a database and fetch records.
- Call an external RESTful Service Provider using the HTTP client.

Prerequisites

- Download and install curl. (This is a command-line tool that you will use to send HTTP requests to the Web Service.)
- (Optional) Download and install H2 Database on your computer.

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. Develop a Web Project with Apache Camel

The purpose of this exercise is to use the Apache CXF framework to expose a Web Service and generate the Java stub code from the WSDL contract. The exposed Web Service will be used by an Apache Camel route while invoking a Bean that contains business logic and populates the response to the HTTP client. The `camel-webservice` project contains the WSDL contract, the Bean definition, and the curl requests.

In this lab exercise, you will complete the following:

- Extend the Maven `pom.xml` file so that it uses the CXF plug-in in this project.
- Design the Apache Camel route so that it exposes a Web Service and dispatches the SOAP requests to the corresponding Bean method.
- Use the SOAP format (with JAXB annotations) to marshall and unmarshall the message to and from Java classes.

In this scenario, the Apache Camel route is defined as follows:

```
from(CXF_WEB_SERVICE).define_Exchange_Pattern_to_return_a_response
  .unmarshall().usingSOAPDataFormat
  .choiceBasedOnSoapAction
    .toBeanMethod("saveCustomer").marshalResponse()
    .ORtoBeanMethod("getCustomerByName").marshalResponse()
    .ORtoBeanMethod("getCustomers").marshalResponse()
```

Follow these steps to complete the exercise:

1. Extend the Maven `pom.xml` file:
   a. In JBoss Developer Studio, use **Project Explorer** to open the `camel-webservice` project.
   b. Review the project contents:
      - com.redhat.gpe.training.camel.BeanService
      - `pom.xml` file
      - WSDL files
   c. Add the CXF plug-in reference to the POM file.

d. Assign the location of the `customerService.wsdl` file as a parameter for the plug-in.

> This plug-in is responsible for generating JAXB annotated classes and Java classes containing the `@WebService` and `@WebMethod` annotations, which the endpoint will use to expose the Web Service and consume incoming HTTP SOAP requests.

e. Run the `mvn compile` command.

f. Review the generated code in the `target/generated/src/main/java/` directory.

2. Define the Apache Camel CXF endpoint, which will handle the HTTP SOAP requests:

a. Open the file `src/main/resources/META-INF/spring/spring-camel-context.xml`.

b. Add the bean definition `<cxf:cxfEndpoint>`.

c. Set the `serviceClass` attribute of the CXF Endpoint definition to `com.redhat.gpe.training.CustomerService`.

d. Set the `address` attribute of the same endpoint to the service's URL: `http://0.0.0.0:9090/training/WebService`.

e. Set the value of the `cxfEndpoint id` to `WS`. Camel uses this id during endpoint lookups in the Spring Beans repository.

```
<cxf:cxfEndpoint id="WS"
                 address="http://0.0.0.0:9090/training/WebService"
                 serviceClass="com.redhat.gpe.training.CustomerService">
</cxf:cxfEndpoint>
```

3. Design the Apache Camel route to expose the CXF Bean endpoint:

a. Assign a CXF Bean Endpoint to the URI value of the consumer Endpoint of the new route: `<from uri="cxf:bean:WS"/>`

b. Assign the following parameter: `dataFormat=MESSAGE`

c. Enable logging on the CXF Bean endpoint using the following property: `loggingFeatureEnabled`

d. Add the Exchange pattern and specify SOAP as the message format within the `SOAPJAXB` tag. The `contextPath` parameter provides the name of the package containing the classes generated by the Apache CXF plug-in (`com.redhat.gpe.training"`).

e. Develop your content-based router to test the `SOAPAction`, which is received using the simple language, and then calls these `beanService` methods: `saveCustomer`, `getCustomerByName`, `getCustomers`

```
<choice>
   <when>
      <simple>${in.header.SOAPAction} contains ...</simple>
   </when>
   ...
 </choice>
```

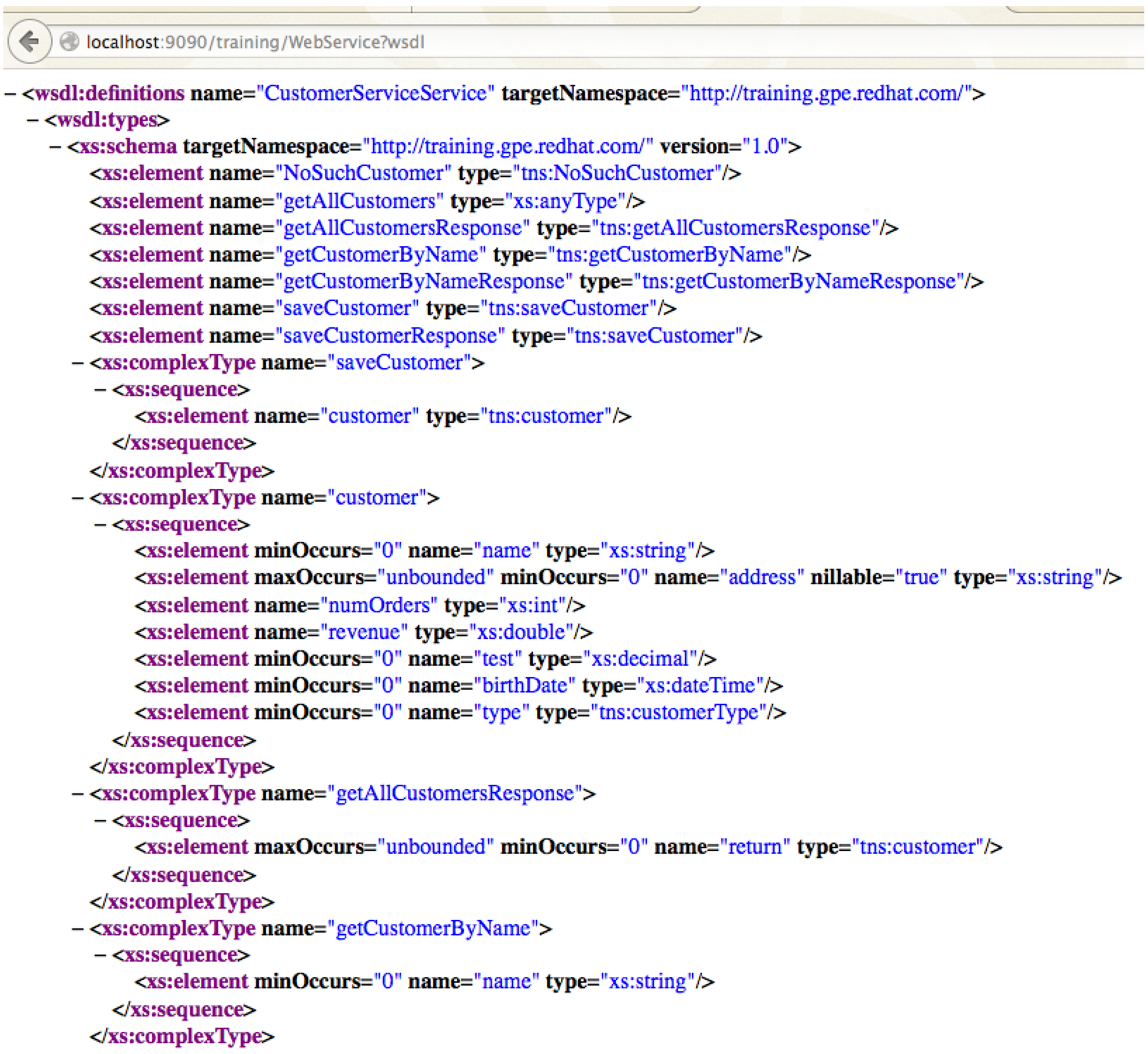
> Remember to format the message before you dispatch it to the CXF endpoint in the `SOAPJAXB` data format. Additionally, you can enrich your Apache Camel Route with the `<log message=""/>` tag, which displays the Exchange Body and SOAP Action.

f. Save your project.

4. Check your work:

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

b. Use a web browser to verify that the Web Service is available at this URL: `http://127.0.0.1:9090/training/WebService?wsdl`.



```
– <wsdl:definitions name="CustomerServiceService" targetNamespace="http://training.gpe.redhat.com/">
  – <wsdl:types>
    – <xs:schema targetNamespace="http://training.gpe.redhat.com/" version="1.0">
        <xs:element name="NoSuchCustomer" type="tns:NoSuchCustomer"/>
        <xs:element name="getAllCustomers" type="xs:anyType"/>
        <xs:element name="getAllCustomersResponse" type="tns:getAllCustomersResponse"/>
        <xs:element name="getCustomerByName" type="tns:getCustomerByName"/>
        <xs:element name="getCustomerByNameResponse" type="tns:getCustomerByNameResponse"/>
        <xs:element name="saveCustomer" type="tns:saveCustomer"/>
        <xs:element name="saveCustomerResponse" type="tns:saveCustomer"/>
      – <xs:complexType name="saveCustomer">
        – <xs:sequence>
            <xs:element name="customer" type="tns:customer"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="customer">
        – <xs:sequence>
            <xs:element minOccurs="0" name="name" type="xs:string"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="address" nillable="true" type="xs:string"/>
            <xs:element name="numOrders" type="xs:int"/>
            <xs:element name="revenue" type="xs:double"/>
            <xs:element minOccurs="0" name="test" type="xs:decimal"/>
            <xs:element minOccurs="0" name="birthDate" type="xs:dateTime"/>
            <xs:element minOccurs="0" name="type" type="tns:customerType"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="getAllCustomersResponse">
        – <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="return" type="tns:customer"/>
          </xs:sequence>
        </xs:complexType>
      – <xs:complexType name="getCustomerByName">
        – <xs:sequence>
            <xs:element minOccurs="0" name="name" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
```

Figure 1. Sample browser output

c. If you can access the Web Service and obtain the WSDL definition, use a curl request to query the Web Service. The curl requests are available in the `src/main/resources/data/curl_requests.txt` directory.

d. Open a terminal and move to the directory `${camel-webservice-project}/src/main/resources/data`:

```
curl -X POST --header "content-type: text/xml" --header "SOAPAction:http://training.gpe.redhat.com/getCustomerByName"
--data @soap-getCustomerByName.xml http://0.0.0.0:9090/training/WebService
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns3="http://training.gpe.redhat.com/">
    <ns2:Body>
        <ns3:getCustomerByNameResponse>
            <return>
                <name>redhat</name>
                <address>FuseSource Office</address>
                <numOrders>85</numOrders>
                <revenue>4015.0</revenue>
                <test>100.0</test>
                <type>BUSINESS</type>
            </return>
        </ns3:getCustomerByNameResponse>
    </ns2:Body>
</ns2:Envelope>

curl -X POST --header "content-type: text/xml" --header "SOAPAction:http://training.gpe.redhat.com/saveCustomer" --
data @soap-saveCustomer.xml http://0.0.0.0:9090/training/WebService

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns3="http://training.gpe.redhat.com/">
    <ns2:Body>
        <ns3:customer>
            <name>Foo</name>
            <address>my address 123</address>
            <numOrders>7</numOrders>
            <revenue>3217.0</revenue>
            <test>100.0</test>
            <type>PRIVATE</type>
        </ns3:customer>
    </ns2:Body>
</ns2:Envelope>

curl -X POST --header "content-type: text/xml" --header "SOAPAction:http://training.gpe.redhat.com/getAllCustomers" -
-data @soap-getAllCustomers.xml http://0.0.0.0:9090/training/WebService

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:Envelope xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns3="http://training.gpe.redhat.com/">
    <ns2:Body>
        <ns3:getAllCustomersResponse>
            <return>
                <name>redhat</name>
                <address>FuseSource Office</address>
                <numOrders>85</numOrders>
                <revenue>4015.0</revenue>
                <test>100.0</test>
                <type>BUSINESS</type>
            </return>
            <return>
                <name>Foo</name>
                <address>my address 123</address>
                <numOrders>7</numOrders>
                <revenue>3217.0</revenue>
                <test>100.0</test>
                <type>PRIVATE</type>
            </return>
        </ns3:getAllCustomersResponse>
    </ns2:Body>
</ns2:Envelope>
```

1.2. Poll a Database Using the SQL Component

In this exercise, you will learn how to use and configure the SQL component which is designed using the Spring Java Database Connectivity (JDBC) framework. Additionally, the SQL component uses the JDBCTemplate class to handle database connections and execute SQL queries (in other words CRUD).

For this exercise, you will use an InMemory Database, which is built in Java and called H2Database. You will create and populate simple tables and fetch and log the data. Additionally, you will create two Apache Camel routes: one to fetch and poll the data and the other to insert flat-file records into the database.

Follow these steps to complete the exercise:

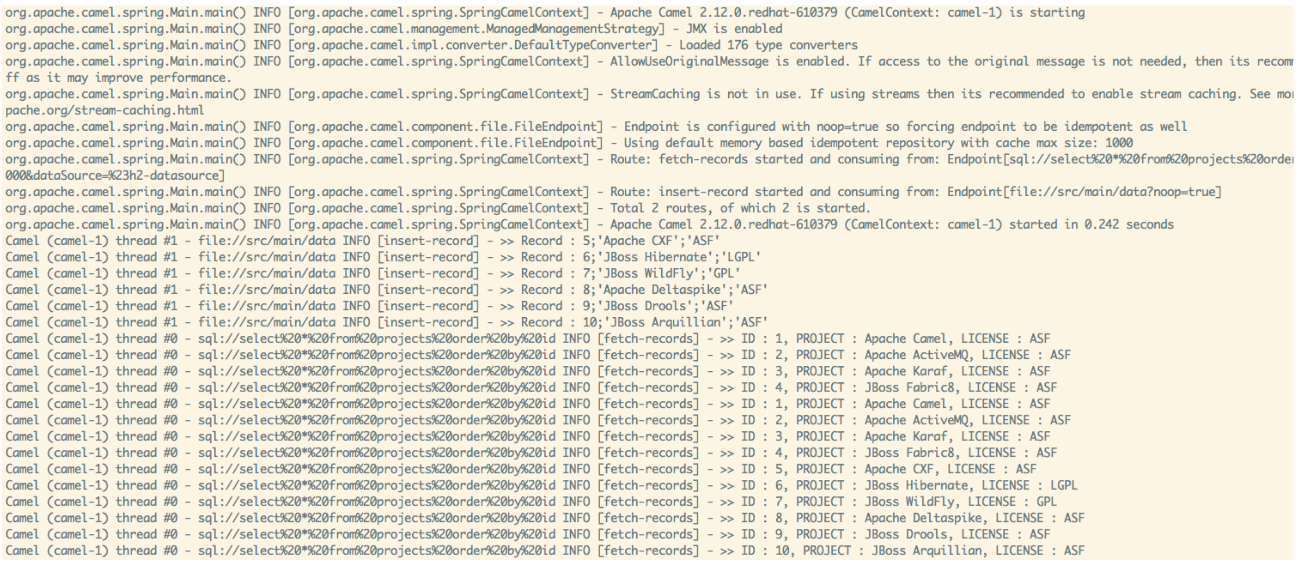
1. Explore the `camel-sql` project:
   a. In JBoss Developer Studio, use **Project Explorer** to open the `camel-sql` project.
   b. Review the project contents:
      - `camel-context.xml`
      - `spring-database.xml`
      - SQL scripts
      - `sql-records.txt`
   c. Observe that the `spring-database.xml` file contains the definition for the `jdbc:embedded-database` bean, which is required to start the H2 database and insert records into the database.
2. Design the first Apache Camel route, which will poll for records from the database:
   a. In the `src/main/resources/META-INF/spring/camel-context.xml` file, locate the `<camelContext/>` bean tag.
   b. Add the `<from uri=""/>` consumer Endpoint and configure the SQL component so that it selects the records from the `projects` table `sql:select * from projects order by id`.
   c. Pass a parameter to the Endpoint specifying the location of the `datasource` bean.
   d. Pass another parameter to the Endpoint indicating that you want to poll the table every `5` seconds.
   e. Because the SQL component returns a `List<?>+ of +Map<String, Object>` object, you must split the content using the `<simple>` language.
   f. The result of the `<split/>` is logged using the Log processor `<log/>`. The message must contain the values: `project`, `id`, and `license`, where each row is a `Map<String, Object>`.

      Refer to the JBoss Fuse documentation to see how simple language extracts this information from the `Map`.

   g. Confirm that your result looks like this: `<log message=">> ID : TODO, PROJECT : TODO, LICENSE : TODO/>`.
3. Design the second Apache Camel Route, which inserts new records into the database using the data from the polled directory:
   a. In the `src/main/resources/META-INF/spring/camel-context.xml` file, locate the `<camelContext/>` bean tag.
   b. Add a `<from uri=""/>` tag to enable the `file` component in the `src/main/data` directory to poll files.
   c. Add an attribute which allows the directory to be polled only once without deleting the contents of the directory.
   d. Convert the body from a `(+GenericFile+ object)` to a String by using the `TypeConversion` strategy.
   e. Use the carriage return char `\n` to split the content, which is currently a list of records (`+7;'JBoss WildFly';'GPL'`).
   f. Use the SQL producer component to insert the content of the record into the database: `sql:insert into projects (id, project, license) values (#, #, #)`

      Check the SQL component documentation for details on passing parameters.

4. Check your work:
   a. Launch the project using `mvn camel:run`.
   b. Use the log in the console to verify that achieved the objective of this exercise.



```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) is starting
org.apache.camel.spring.Main.main() INFO [org.apache.camel.management.ManagedManagementStrategy] - JMX is enabled
org.apache.camel.spring.Main.main() INFO [org.apache.camel.impl.converter.DefaultTypeConverter] - Loaded 176 type converters
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - AllowUseOriginalMessage is enabled. If access to the original message is not needed, then its recomm
ff as it may improve performance.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - StreamCaching is not in use. If using streams then its recommended to enable stream caching. See mo
pache.org/stream-caching.html
org.apache.camel.spring.Main.main() INFO [org.apache.camel.component.file.FileEndpoint] - Endpoint is configured with noop=true so forcing endpoint to be idempotent as well
org.apache.camel.spring.Main.main() INFO [org.apache.camel.component.file.FileEndpoint] - Using default memory based idempotent repository with cache max size: 1000
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: fetch-records started and consuming from: Endpoint[sql://select%20*%20from%20projects%20orde
000&dataSource=%23h2-datasource]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: insert-record started and consuming from: Endpoint[file://src/main/data?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 0.242 seconds
Camel (camel-1) thread #1 - file://src/main/data INFO [insert-record] - >> Record : 5;'Apache CXF';'ASF'
Camel (camel-1) thread #1 - file://src/main/data INFO [insert-record] - >> Record : 6;'JBoss Hibernate';'LGPL'
Camel (camel-1) thread #1 - file://src/main/data INFO [insert-record] - >> Record : 7;'JBoss WildFly';'GPL'
Camel (camel-1) thread #1 - file://src/main/data INFO [insert-record] - >> Record : 8;'Apache Deltaspike';'ASF'
Camel (camel-1) thread #1 - file://src/main/data INFO [insert-record] - >> Record : 9;'JBoss Drools';'ASF'
Camel (camel-1) thread #1 - file://src/main/data INFO [insert-record] - >> Record : 10;'JBoss Arquillian';'ASF'
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 1, PROJECT : Apache Camel, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 2, PROJECT : Apache ActiveMQ, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 3, PROJECT : Apache Karaf, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 4, PROJECT : JBoss Fabric8, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 1, PROJECT : Apache Camel, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 2, PROJECT : Apache ActiveMQ, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 3, PROJECT : Apache Karaf, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 4, PROJECT : JBoss Fabric8, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 5, PROJECT : Apache CXF, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 6, PROJECT : JBoss Hibernate, LICENSE : LGPL
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 7, PROJECT : JBoss WildFly, LICENSE : GPL
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 8, PROJECT : Apache Deltaspike, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 9, PROJECT : JBoss Drools, LICENSE : ASF
Camel (camel-1) thread #0 - sql://select%20*%20from%20projects%20order%20by%20id INFO [fetch-records] - >> ID : 10, PROJECT : JBoss Arquillian, LICENSE : ASF
```

Figure 2. SQL results

1.3. Call an External RESTful Service Provider Using the HTTP Client

The goal of this exercise is to call a RESTFul Service that interfaces with the Marvel Comics Database and retrieves information regarding comic superheroes. You will poll a directory for a file that contains a delimited list of identifiers for the comic superheroes that you want to retrieve. Next you will prepare the HTTP requests for each web service call, and then you will call the RESTful service `http://gateway.marvel.com/v1/public/characters/id`. To accomplish this, you will create two Apache Camel routes: one route to poll the directory and a second route to call the RESTful service.

> You can find more info about this web service here and in the official Marvel Developer Documentation.

Follow these steps to complete the exercise:

1. Explore the `camel-rest` project:
   a. In JBoss Developer Studio, use **Project Explorer** to open the `camel-sql` project.
   b. Review the project contents:
      - `com.redhat.gpe.training.camel.MarvelUtil`
      - Model classes
2. Set data format properties:
   a. Open the file `src/main/resources/META-INF/spring/camel-context.xml`
   b. Under the `<camelContext/>` tag, add the specify `json` as the data format and `jackson` as the library you want to use.
   c. Set the `unmarshalTypeName` attribute to pass the FQN for the `Record` class defined in the `model` package. This class corresponds to the root class that Jackson will map with the `JSONResponse` object returned by the RESTFul service.

      > The component properties are defined under the `<camelContext/>` tag element and are used to obtain the values of the keys declared in the `marvel.properties` file, such as the API key and the private key.
3. Create the first route. This route polls a directory `file://src/main/data?noop=true` for a file containing a list of IDs that are separated by a comma character. The content must be split using this token char `token=","`. Each new Exchange that contains a Marvel ID String will be used to call a `direct:call-marvel` Endpoint. Before calling the Endpoint, you must set different headers to configure the `HTTP Query`, `HTTP URI`, and `HTTP Method`. To accomplish this, follow these steps:
   a. Add a `<setHeader/>` tag element where the header name is `hash` and the value retrieved from the `hash` header is defined within the `marvelUtil` bean. A hash String is calculated and sent as a required parameter by Marvel.

b. Add another header called `ts` where you get the `ts` object from the `marvelUtil` bean. The `ts` field returns a System Timestamp, which is also required by the Marvel RESTful web service.

c. Add a third header called `marvelID` that uses the `Body` content as the value.

d. Set the `CamelHttpQuery` header to use the value derived from the calculation performed by the following simple expression:
`<simple>hash=${header.hash}&amp;apikey=${properties:apiKey}&amp;ts=${header.ts}</simple>`

e. Set the `CamelHttpUri` header to query the following URI using the `simple` language:
`http://gateway.marvel.com/v1/public/characters/nnnn`

   Remember to replace `nnnn` with the value of the `Body` or `marvelID` header.

f. Set the `CamelHttpMethod` header with the value `GET` using the `constant` expression language.

g. Call the `direct:call-marvel` endpoint. The first route is now complete.

4. Define the second route. This route starts with the `from("direct:call-marvel")` endpoint, which dispatches an Exchange to an HTTP endpoint. The result received by the HTTP endpoint is unmarshalled using the JSON data format. The POJO instantiated is used to log the information received. To accomplish this, follow these steps:

   a. Create the second route and add the `<from/>` tag element where the URI definition corresponds to `direct:call-marvel`.

   b. Add the `<to uri=""/>` tag where the address of the service is specified as `http://gateway.marvel.com/v1/public/`.

   c. Specify the format to which the message will be unmarshalled: `<unmarshal ref=""/>`. For example: JSON, SOAP, Bindy, String.

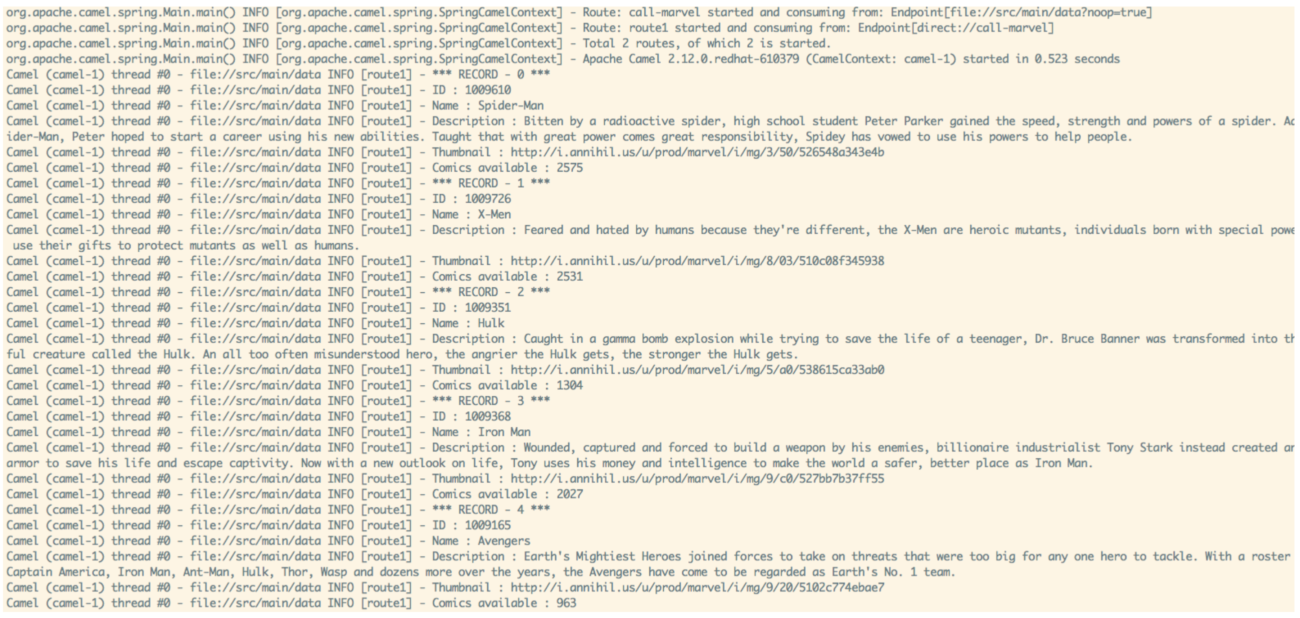
   d. Log the message received using the `<log message="xxxxx : ${yyyyy}"/>` format, where `xxxxx` is one of the attributes listed below, and `yyyyy` is the value contained in the header or body object expressed using the simple language:

      - ID
      - Name
      - Description
      - Thumbnail
      - Comics available

5. Check your work:

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

   b. Use the console log to validate the message response:



```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: call-marvel started and consuming from: Endpoint[file://src/main/data?noop=true]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: route1 started and consuming from: Endpoint[direct://call-marvel]
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 0.523 seconds
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - *** RECORD - 0 ***
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - ID : 1009610
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Name : Spider-Man
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Description : Bitten by a radioactive spider, high school student Peter Parker gained the speed, strength and powers of a spider. Ac
ider-Man, Peter hoped to start a career using his new abilities. Taught that with great power comes great responsibility, Spidey has vowed to use his powers to help people.
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Thumbnail : http://i.annihil.us/u/prod/marvel/i/mg/3/50/526548a343e4b
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Comics available : 2575
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - *** RECORD - 1 ***
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - ID : 1009726
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Name : X-Men
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Description : Feared and hated by humans because they're different, the X-Men are heroic mutants, individuals born with special powe
 use their gifts to protect mutants as well as humans.
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Thumbnail : http://i.annihil.us/u/prod/marvel/i/mg/8/03/510c08f345938
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Comics available : 2531
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - *** RECORD - 2 ***
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - ID : 1009351
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Name : Hulk
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Description : Caught in a gamma bomb explosion while trying to save the life of a teenager, Dr. Bruce Banner was transformed into th
ful creature called the Hulk. An all too often misunderstood hero, the angrier the Hulk gets, the stronger the Hulk gets.
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Thumbnail : http://i.annihil.us/u/prod/marvel/i/mg/5/a0/538615ca33ab0
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Comics available : 1304
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - *** RECORD - 3 ***
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - ID : 1009368
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Name : Iron Man
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Description : Wounded, captured and forced to build a weapon by his enemies, billionaire industrialist Tony Stark instead created ar
armor to save his life and escape captivity. Now with a new outlook on life, Tony uses his money and intelligence to make the world a safer, better place as Iron Man.
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Thumbnail : http://i.annihil.us/u/prod/marvel/i/mg/9/c0/527bb7b37ff55
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Comics available : 2027
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - *** RECORD - 4 ***
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - ID : 1009165
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Name : Avengers
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Description : Earth's Mightiest Heroes joined forces to take on threats that were too big for any one hero to tackle. With a roster
Captain America, Iron Man, Ant-Man, Hulk, Thor, Wasp and dozens more over the years, the Avengers have come to be regarded as Earth's No. 1 team.
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Thumbnail : http://i.annihil.us/u/prod/marvel/i/mg/9/20/5102c774ebae7
Camel (camel-1) thread #0 - file://src/main/data INFO [route1] - Comics available : 963
```

Figure 3. Response received

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