Table of Contents

Business Scenario

1. Import the Lab Project
   - 1.1. Build the `parent` Project
   - 1.2. Build the `artifacts` Project

2. Develop the Inbound Application
   - 2.1. CXF REST Service
   - 2.2. Marshal the Java Object to XML
   - 2.3. Publish the XML on a Red Hat AMQ queue
   - 2.4. Develop the JUnit Tests
   - 2.5. Build the `Inbound` Project

3. Develop the Xlate Application
   - 3.1. Consume the Message from Red Hat AMQ
   - 3.2. Unmarshall the XML to a Java Object
   - 3.3. TypeConvert to a Class that Matches `ExecuteMatchUpdate`
   - 3.4. Marshall to XML
   - 3.5. Publish to Red Hat AMQ
   - 3.6. Build the `Xlate` Project

4. Develop the Outbound Application
   - 4.1. Consume the Message from Red Hat AMQ
   - 4.2. Marshal the Java Object to XML
   - 4.3. Invoke the Nextgate Web Service
   - 4.4. Develop the JUnit Tests
   - 4.5. Deploy the Test Web Service in Apache Karaf
   - 4.6. Build the `Outbound` Project

5. Grading criteria

Business Scenario

This project facilitates synchronization of master patient records across different healthcare providers. It is important to have consistent patient information across these multiple providers so that patients may receive consistent care. For that to occur, their personal and medical information needs to be shared. Updates to the patient record also need to flow across the providers to maintain accuracy and currency.

Technical goals to accomplish the above:

- Build a RESTful CXF service that receives the patient record and validates it.

- Publish the XML to a queue and send an acknowledgement back to the client (Done transformer in the use case document).
- Receive the message from the messaging system and invoke the backend web service. The Nextgate web service will be provided for testing. Respond to the client using the Done transformer.

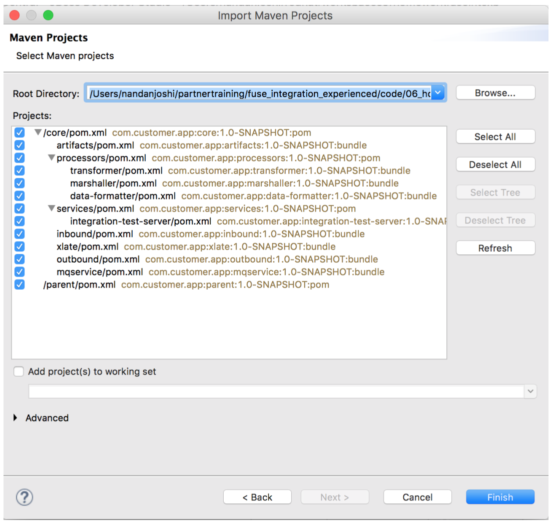
---

1. Import the Lab Project

1. In Red Hat Developer Studio, switch to the **Git** perspective.
2. Click the icon at the top to clone a Git repository and add the clone to this view.
3. In the **URI** text box, copy and paste the following:

```
https://github.com/gpe-mw-training/experienced-integration-
labs.git
```

4. Switch to **Project Explorer** for the **JBoss** perspective.
5. Import a new Maven project by selecting **File** → **Import** → **Maven** → **Existing Maven Projects** → **Next**. Import the `core` and its parent project.



Import Maven Projects

Maven Projects
Select Maven projects

Root Directory: /Users/nandanjoshi/partnertraining/fuse_integration_experienced/code/06_hc

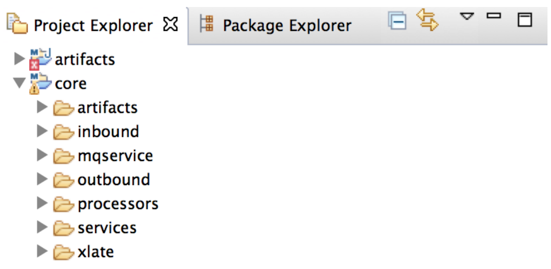
Projects:

```
☑ ▼/core/pom.xml  com.customer.app:core:1.0-SNAPSHOT:pom
☑     artifacts/pom.xml  com.customer.app:artifacts:1.0-SNAPSHOT:bundle
☑   ▼processors/pom.xml  com.customer.app:processors:1.0-SNAPSHOT:pom
☑       transformer/pom.xml  com.customer.app:transformer:1.0-SNAPSHOT:bundle
☑       marshaller/pom.xml  com.customer.app:marshaller:1.0-SNAPSHOT:bundle
☑       data-formatter/pom.xml  com.customer.app:data-formatter:1.0-SNAPSHOT:bundle
☑   ▼services/pom.xml  com.customer.app:services:1.0-SNAPSHOT:pom
☑       integration-test-server/pom.xml  com.customer.app:integration-test-server:1.0-SNAI
☑     inbound/pom.xml  com.customer.app:inbound:1.0-SNAPSHOT:bundle
☑     xlate/pom.xml  com.customer.app:xlate:1.0-SNAPSHOT:bundle
☑     outbound/pom.xml  com.customer.app:outbound:1.0-SNAPSHOT:bundle
☑     mqservice/pom.xml  com.customer.app:mqservice:1.0-SNAPSHOT:bundle
☑  /parent/pom.xml  com.customer.app:parent:1.0-SNAPSHOT:pom
```

☐ Add project(s) to working set

Figure 1. Homework project

6. Click the **Project Explorer** tab and expand the `Inbound` project node.



```
Project Explorer   Package Explorer
▶ artifacts
▼ core
  ▶ artifacts
  ▶ inbound
  ▶ mqservice
  ▶ outbound
  ▶ processors
  ▶ services
  ▶ xlate
```

Figure 2. Application structure

7. Examine the following files and folders that appear in the expanded view:
   - `core`: Area in which you will develop the three parts of this use case.

Sub-folders are as follows:

- `artifacts`: WSDL and XSD files you work from
- `inbound`: Route or service that receives the patient payload
- `xlate`: Route that marshalls the Java object to XML
- `outbound`: Route that publishes the XML payload on a A-MQ queue

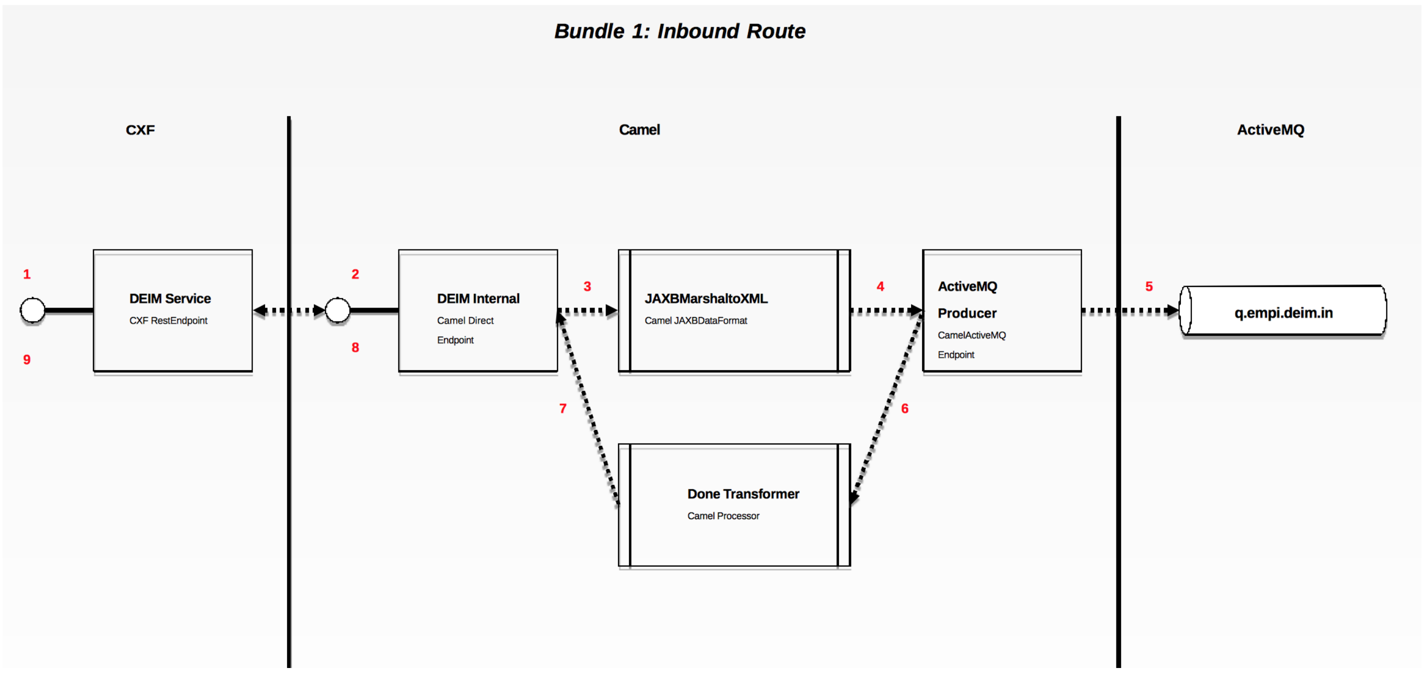
1.1. Build the `parent` Project

1. On the command line, run `mvn clean install`.

1.2. Build the `artifacts` Project

1. On the command line, run `mvn clean install`. Among other things, it generates the Java client/server stubs from the WSDL files.

2. Develop the Inbound Application



2.1. CXF REST Service

For this section, we recommend that you use a standard REST service CXF-RS. You can use Spring or Blueprint which starts a RESTful service with an implementation. Inside the implementation, you can create a producer template and call a direct endpoint in Camel. Go to `blueprint.xml` and create a bean, which is the REST service. You will implement the add, update, and search methods in the REST endpoint service.

Alternately, you can use the Camel REST component.

1. Use the former design to formulate a response to the client that indicates why the request failed. Also you can validate schema against the incoming payloads.
2. Implement a Java class that is the REST endpoint.

- The class creates a producer template and calls the direct route.
- The direct route puts the message on A-MQ and returns, allowing you to return an *OK* status to the client in the Java class.

2.2. Marshal the Java Object to XML

1. Using the JAXB marshaller, convert the payload to XML.

2.3. Publish the XML on a Red Hat AMQ queue

Once the message is successfully published, a *Done* XML string is sent back to the client.

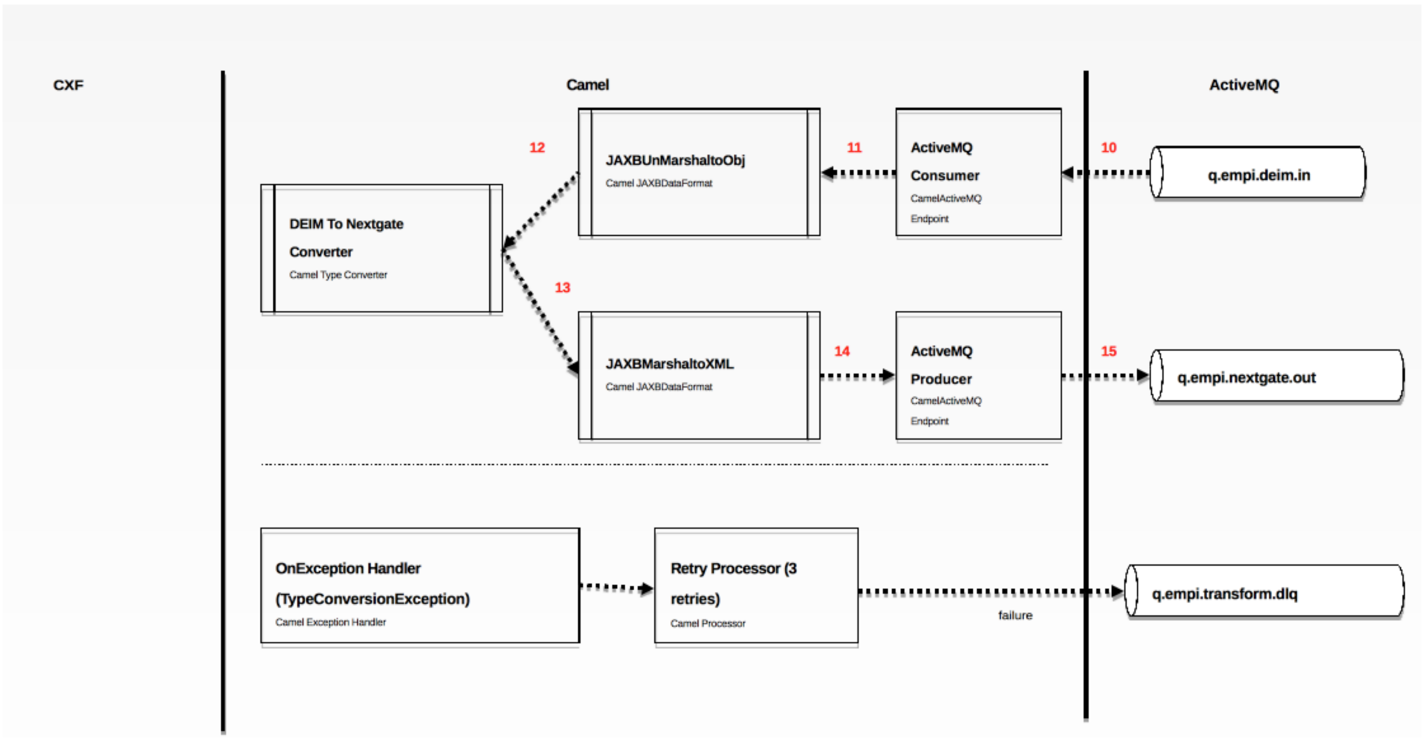
2.4. Develop the JUnit Tests

1. Develop a feature and a fabric profile.
2. Test these using SoapUI or curl.

2.5. Build the `Inbound` Project

1. On the command line, run `mvn clean install`.

3. Develop the Xlate Application



3.1. Consume the Message from Red Hat AMQ

This consumes the Person XML from an ActiveMQ queue (`q.empi.deim.in`).

3.2. Unmarshall the XML to a Java Object

This converts the Person XML to a Person object.

3.3. TypeConvert to a Class that Matches `ExecuteMatchUpdate`

This converts the Person object to an `ExecuteMatchUpdate` object, which is the operation on the web service. To accomplish this, you have to write your own TypeConverter with the `convertTo` method implemented. The parameters to this method are as follows:

- Type: The type of the object you want to convert to
- Exchange: This is passed in. Executing a `getIn()` or `getBody()` returns a Person object that can be copied into the object.
- Object: The object you want to convert to.

3.4. Marshall to XML

This converts the `ExecuteMatchUpdate` object to `ExecuteMatchUpdate` XML.

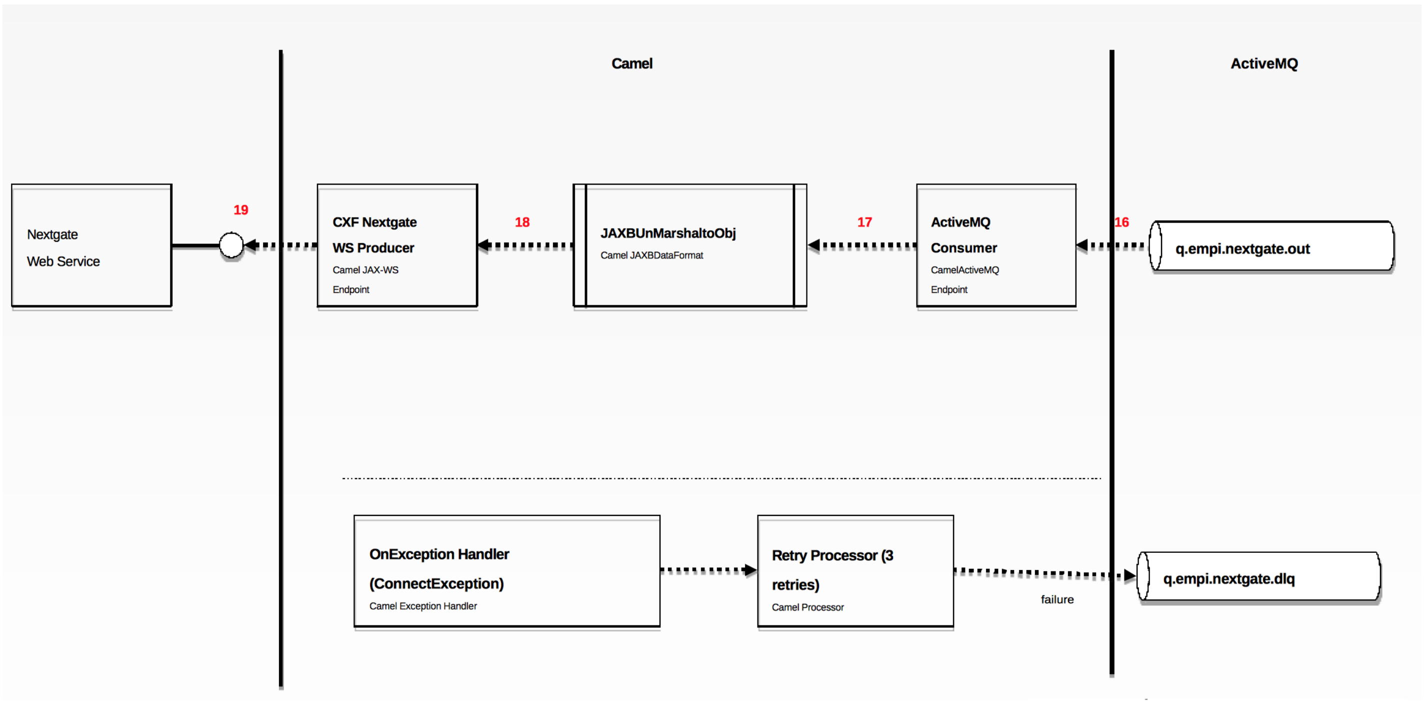
3.5. Publish to Red Hat AMQ

This dispatches the `ExecuteMatchUpdate` XML to an ActiveMQ queue (`q.empi.nextgate.out`).

3.6. Build the `Xlate` Project

1. On the command line, run `mvn clean install`.

4. Develop the Outbound Application



4.1. Consume the Message from Red Hat AMQ

This consumes the `ExecuteMatchUpdate` XML from an ActiveMQ queue (`q.empi.nextgate.out`).

4.2. Marshal the Java Object to XML

This converts the `ExecuteMatchUpdate` XML to an `ExecuteMatchUpdate` object.

4.3. Invoke the Nextgate Web Service

1. Use the `ExecuteMatchUpdate` object to build the Nextgate WebService Request (sent to WebService).
   - This dispatches the request to the Nextgate service URI (`CXF_WS_ENDPOINT_URI`).
   - The input is the WSDL for the web service.

4.4. Develop the JUnit Tests

1. Develop a feature and a fabric profile.
2. Test these using SoapUI or curl.

4.5. Deploy the Test Web Service in Apache Karaf

1. Build the `integration-test-server` JAR in the `services/integration-test-server` folder. This project depends on the local artifact `osgi:install mvn:com.customer.app/artifacts/1.0-SNAPSHOT`.

   Compile the core project to get this JAR.

- % bin/karaf
- >osgi:install mvn:com.customer.app/integration-test-server/1.0-SNAPSHOT
- >osgi:install mvn:com.customer.app/artifacts/1.0-SNAPSHOT
- >osgi:start [bundle-id for integration-test-server]
- >osgi:start [bundle-id for artifacts]
- >osgi:list

4.6. Build the `Outbound` Project

1. On the command line, run `mvn clean install`.

5. Grading criteria

1. 5 points for building out the routes in all the sub-projects
2. 5 points for clear instructions on how to build and run the homework assignment
3. 10 points if the individual sub-projects build cleanly and their JUnit tests run without errors (20 points is the passing grade)
4. 10 points for building out a significant set of tests for the scenario or creating a significant dataset and testing using `mvn camel:run`
5. 5 points for a concise explanation of the design choices made, and areas for improvement

Build Version: 1.0R : Last updated 2019-03-04 09:44:50 EST