FuseSource

Hands on
with Apache ActiveMQ,
ServiceMix, Camel and CXF

Charles Mouillard and
Gert Vanthienen

DEVOXX™
the java™ community conference

FuseSource

FuseSource, a wholly owned subsidiary of Progress Software, is a community of open source experts that provide software, support, training, and consulting for the most popular Apache-licensed open source integration projects including Apache ServiceMix, ActiveMQ, Camel and CXF. The FuseSource team includes key committers and the leaders at Apache who know the code the best to help FuseSource customers build reliable and scalable software integration infrastructure.

Contact FuseSource

FuseSource,
14 Oak Park,
Bedford, MA 01730 USA

On the Web: fusesource.com
Email: info@fusesource.com

Toll Free: 888.280.5380
Tel: 781.382.4002
Outside US: +31 10 235 11 22

INTRODUCTION

The goal of this Devoxx "Hands-On Lab"is to show how you can develop a Java Integration project using Apache Technologies such as Apache Camel, ActiveMQ, CXF and ServiceMix. The project that we will develop is a simple application containing a persistence layer, a web interface, a web service and Camel to put the glue between the different components as depicted here.
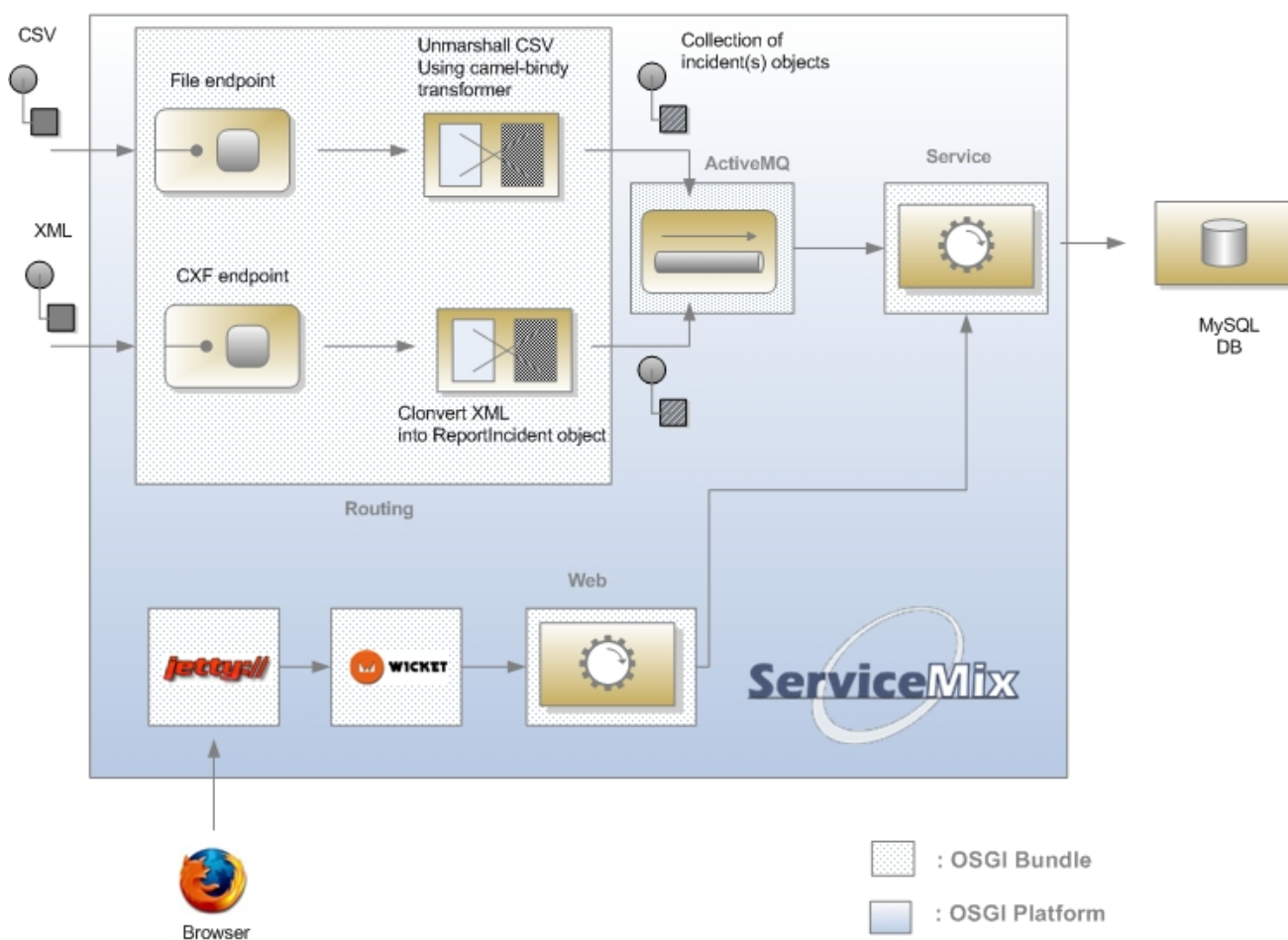
The application is listening for incidents coming from web service or files. According to the origin, the content (= incidents) are transformed into their corresponding objects using for the CSV file, a data formatter engine called camel-bindy and for the Web Service camel-cxf component. Each message transformed is placed in a queue handled by ActiveMQ engine. All the messages (containing the objects) are next processed by a Bean service that will (with the help of injection of dependency provided by Spring) save the incidents in a H2 Database using Spring and OpenJPA frameworks.

A small Apache Wicket web application running in Jetty Web server provides to the users a screen to consult the incidents created.

Project structure

The project has been cut into the following components (maven modules) :

Maven project name	Description
db	Contain script to create Schema and database for H2
features	features provisioning file containing the bundles
model	model layer
persistence-jpa	JPA persistence layer
routing	camel routes
service	spring service layer
web	Apache wicket module
webservice	Apache CXF web service

Each layer of our application will be deployed as separate bundle into Fuse ESB container

Bundle = JAR file where the `META-INF/MANIFEST.MF` file has been modified to include the OSGi meta-data (packages to be imported, exported, ...)

Prerequisites

- JDK 6.x or 7.x

- Apache Maven 3.0.3 must be installed and configured

- Eclipse Indigo 3.7.1 or Fuse IDE

- Fuse ESB

- H2 Database

- SOAPUI

Except for Apache Maven and JDK, the other products are available here : http://192.168.1.2/~charlesmoulliard/software/

Download and install the project skeleton

To simplify the setup of the project, we provide you a maven modules structure. It is available here : http://192.168.1.2/~charlesmoulliard/software/fuse-camel-integration.zip

STEP 1 : Unzip it under `~/devoxx/` or `C:\devoxx`

STEP 2 : Start a Dos / Unix console

STEP 3 : `cd ~/devoxx/fuse-camel-integration` or `D:\devoxx\fuse-camel-integration`

STEP 4 : execute `mvn clean install` to verify that maven works

MODEL LAYER

As our application receive Incident report, we will create a java model class to support the information that we will receive and will additionally use it to persist info into the database using JPA specification.

The reportincident model is really simple because it only contains one class that we will use :

- to map information with the database, CSV file,

- to transport information to web screen.

STEP 1 : Move to to the directory/maven module `persistence-jpa`

STEP 2 : Create under `src/main/java/org/fusesource/devoxx/reportincident/model` directory the `Abstract` and `Incident` class and add the fields that will map info that we receive

Remark : The `pom.xml` file of each module contains all the dependencies required.

```java
package org.fusesource.devoxx.reportincident.model;

import java.util.Date;

public class Incident extends Abstract implements Serializable {

    private static final long serialVersionUID = 1L;

    private String incidentRef;
    private Date incidentDate;
    private String givenName;
    private String familyName;
    private String summary;
    private String details;
    private String email;
    private String phone;
    private long incidentId;
    private String creationUser;
    private Date creationDate;

}
```

STEP 2 : Generate the Getter and Setter fields
STEP 3 : Add camel–bindy annotations
Bindy (http://camel.apache.org/bindy.html) is one of the data format supported by Camel like JAXB, SOAP, CSV, SMOOKS, Dozer, JSON, Castor, Xstreams, XmlBeans, Google Protobuf (http://camel.apache.org/data-format.html). It will allow mapping the CSV record field with the java properties using Java @Annotations as we do with JAXB

Add the following annotations in the `Incident` class

```java
import org.apache.camel.dataformat.bindy.annotation.CsvRecord;
import org.apache.camel.dataformat.bindy.annotation.DataField;

@CsvRecord(separator = ",")
public class Incident extends Abstract implements Serializable {

    private static final long serialVersionUID = 1L;

    @DataField(pos = 1)
    private String incidentRef;

    @DataField(pos = 2, pattern = "dd-mm-yyyy")
    private Date incidentDate;

    @DataField(pos = 3)
    private String givenName;

    @DataField(pos = 4)
    private String familyName;

    @DataField(pos = 5)
    private String summary;

    @DataField(pos = 6)
    private String details;

    @DataField(pos = 7)
    private String email;

    @DataField(pos = 8)
    private String phone;

    /* SNIP */
}
```

STEP 4 : The Incident class extend Abstract class which overidde toString method using Apache Commons Lang to build easily display object content when we trace Apache Camel route

```java
package org.fusesource.devoxx.reportincident.model;

import org.apache.commons.lang.builder.ToStringBuilder;
import org.apache.commons.lang.builder.ToStringStyle;

public abstract class Abstract {

    @Override
    public String toString() {
        return ToStringBuilder.reflectionToString(this,
                                            ToStringStyle.MULTI_LINE_STYLE);
    }

}
```

Extend model layer to support JPA

Our Incident model class will also be used as an Entity class to insert Report Incident in the database. We will use the JPA specification for that purpose and Apache OpenJPA (http://openjpa.apache.org/builds/2.1.1/apache-openjpa/docs/manual.html).

Our class/Entity will be mapped with the `T_INCIDENT` table, the definition of the different columns, their name and fields length are provided here after

STEP 1 : Add JPA annotations `@Entity`, `@Table`, `@Column`

```java
import javax.persistence.*;

@Entity
@Table(name = "T_INCIDENT")
public class Incident extends Abstract implements Serializable {

    @Column(name = "INCIDENT_REF", length = 55)
    private String incidentRef;

    @Column(name = "INCIDENT_DATE")
    private Date incidentDate;

    @Column(name = "GIVEN_NAME", length = 35)
    private String givenName;

    @Column(name = "FAMILY_NAME", length = 35)
    private String familyName;

    @Column(name = "SUMMARY", length = 35)
    private String summary;

    @Column(name = "DETAILS")
    private String details;

    @Column(name = "EMAIL", length = 60)
    private String email;

    @Column(name = "PHONE", length = 35)
    private String phone;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "INCIDENT_ID")
    private long incidentId;

    @Column(name = "CREATION_USER")
    private String creationUser;

    @Column(name = "CREATION_DATE")
    private Date creationDate;

    /* SNIP */
}
```

As the Entity model is defined, we will now configure the persistence file that the JPA container will use

STEP 2 : Edit the persistence file in the directory `src/main/resources/META-INF/persistence.xml` and define the persistence unit name, the class to be persisted and the properties used by the JPA provider

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation=
               "http://java.sun.com/xml/ns/persistence
                 http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">

  <persistence-unit name="reportIncident" transaction-type="RESOURCE_LOCAL">
        <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
        <class>org.fusesource.devoxx.reportincident.model.Incident</class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
            <property name="openjpa.jdbc.SynchronizeMappings"
                      value="buildSchema(SchemaAction='add,deleteTableContents')"/>
            <property name="openjpa.Log"
                      value="DefaultLevel=TRACE, Runtime=TRACE, Tool=TRACE,
                            SQL=TRACE"/>
            <property name="openjpa.jdbc.DBDictionary"
                      value="h2(useSchemaName=true)"/>
            <property name="openjpa.jdbc.Schema"
                      value="REPORT"/>
        </properties>
    </persistence-unit>

</persistence>
```

DEFINE PERSISTENCE LAYER (DAO)

Now that the model exists, we will create the persistence and layer services. The projects have been designed using the pattern Data Access Object because it allows changing the implementation from a database type to another, between ORM very easily.

Moreover interfaces are used as 'contract' between the services and the DAO. This offers the advantage to decouple objects in the application and as you will see later on it will allow us to deploy services, persistence as separate bundles in the OSGI server

First we create the DAO Interface and its implementation

STEP 1 : Create the Interface `IncidentDAO` in the directory `src/main/java/org/fusesource/devoxx/reportincident/dao`

```java
package org.fusesource.devoxx.reportincident.dao;

import java.util.List;
import org.fusesource.devoxx.reportincident.model.Incident;

public interface IncidentDAO {

  public Incident getIncident(long paramLong);
  public List<Incident> findIncident();
  public List<Incident> findIncident(String paramString);
  public void saveIncident(Incident paramIncident);
  public void removeIncident(long paramLong);

}
```

STEP 2 : Its implementation `IncidentDAOImpl`

```java
package org.fusesource.devoxx.reportincident.dao.impl;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import org.fusesource.devoxx.reportincident.dao.IncidentDAO;
import org.fusesource.devoxx.reportincident.model.Incident;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class IncidentDAOImpl implements IncidentDAO {

  private static final transient Log LOG = LogFactory.getLog(IncidentDAOImpl.class);

  @PersistenceContext
  private EntityManager em;
  private static final String findIncidentByReference =
          "select i from Incident as i where i.incidentRef = :ref";
  private static final String findIncident = "select i from Incident as i";

  public List<Incident> findIncident()
  {
    Query q = this.em.createQuery("select i from Incident as i");
    List list = q.getResultList();
    return list;
  }

  public List<Incident> findIncident(String key) {
    Query q =
            this.em.createQuery("select i from Incident as i where i.incidentRef = :ref");
    q.setParameter("ref", key);
    List list = q.getResultList();
    return list;
  }

  public Incident getIncident(long id) {
    return (Incident)this.em.find(Incident.class, Long.valueOf(id));
  }

  public void removeIncident(long id)  {
    Object record = this.em.find(Incident.class, Long.valueOf(id));
    this.em.remove(record);
    this.em.flush();
  }

  public void saveIncident(Incident incident) {
    this.em.persist(incident);
    this.em.flush();
  }
}
```

To create the Local JPA Container, we will use Spring to simplify our life and also to setup the DataSource and the Transaction Manager required by JPA.

STEP 3 : Edit the file under `src/main/resources/META-INF/spring/persistence-dao.xml`

STEP 4: Add Spring beans to scan `@Persistencecontext` annotation under `<beans>`

```
<context:annotation-config/>
<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>
```

STEP 5 : Define the bean `LocalContainerEntityManagerFactoryBean`

```
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="persistenceUnitName" value="reportIncident"/>
  <property name="jpaVendorAdapter" ref="jpaAdapterH2"/>
  <property name="dataSource" ref="dataSourceH2"/>
</bean>
```

STEP 6 : Define the `OpenJPAVendorAdapter` with H2 properties

```
<bean id="jpaAdapterH2" class="org.springframework.orm.jpa.vendor.OpenJpaVendorAdapter">
  <property name="databasePlatform" value="org.apache.openjpa.jdbc.sql.H2Dictionary"/>
  <property name="database" value="H2"/>
  <property name="showSql" value="true"/>
</bean>
```

STEP 7 : Define the datasource using Apache Commons DBCP

```
<bean id="dataSourceH2" destroy-method="close"
      class="org.apache.commons.dbcp.BasicDataSource" >
  <property name="driverClassName" value="org.h2.Driver"/>
  <property name="url" value="jdbc:h2:tcp://localhost/~/reportdb"/>
  <property name="username" value="sa"/>
  <property name="password" value=""/>
</bean>
```

STEP 8 : As JPA and EntityManager will work with Transactions, a Transaction Manager must be created

```
<!-- TransactionManager is required -->
<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory"/>
  <property name="dataSource" ref="dataSourceH2"/>
</bean>
```

STEP 9 : The Transaction manager created previously will be exported as an OSGI service. By default, an OSGI platform is not JNDI compliant but proposes a mechanism developed by Spring DM () allowing to register a proxy bean using its interface name. In our example we will also using a key define a filter to retrieve from the list of the interfaces registered under org.springframework.transaction.PlatformTransactionManager this one.

```
<!-- Expose Transaction Manager -->
<osgi:service ref="txManager"
              interface="org.springframework.transaction.PlatformTransactionManager">
  <osgi:service-properties>
    <entry key="tx" value="JPA"/>
  </osgi:service-properties>
</osgi:service>
```

STEP 10: Create the DAO bean and register also its interface as an OSGI service which will; be used by the Service layer

```xml
<!-- DAO Declarations -->
<bean id="incidentDAO"
      class="org.fusesource.devoxx.reportincident.dao.impl.IncidentDAOImpl"/>

<!-- Expose DAO interface as OSGI Service -->
<osgi:service ref="incidentDAO"
              interface="org.fusesource.devoxx.reportincident.dao.IncidentDAO"/>
```

SERVICE LAYER

In term of design, the service project is very similar to the persistence because we will create an interface and its implementation. Why repeating the interface. The answer is evident; it is for decoupling the service from the DAO implentation to allow you to switch easily from one ORM to another, ...

STEP 1 : Edit the following file `src/main/java/org/fusesource/devoxx/reportincident/service/` `IncidentService.java` in the module `service`

```java
package org.fusesource.devoxx.reportincident.service;

import java.util.List;
import org.fusesource.devoxx.reportincident.model.Incident;

public interface IncidentService {

  public Incident getIncident( long id );
  public List<Incident> findIncident();
  public List<Incident> findIncident( String key );
  public void saveIncident( Incident incident );
  public void removeIncident( long id );

}
```

STEP 2 : Edit its implementation in the sub-directory `impl/IncidentServiceImpl.java`

```java
package org.fusesource.devoxx.reportincident.service;

import java.util.List;
import org.fusesource.devoxx.reportincident.model.Incident;

public interface IncidentService {

  public Incident getIncident( long id );
  public List<Incident> findIncident();
  public List<Incident> findIncident( String key );
  public void saveIncident( Incident incident );
  public void removeIncident( long id );

}
```

As you see, we will use `IncidentDAO` class within the `IncidentServiceImpl` class. The injection of this bean will be done by Spring. As Spring will be used as Transaction Manager, the service layer will be configured to define which class/methods are transactional and which scope "PROPAGATION_REQUIRED" will be used to check / retrieve the transaction from the transaction manager. We will not use @Transactional annotation but the could be easily transformed to use Spring Transactional Annotations

STEP 3 : Edit the file `src/main/resources/META-INF/spring/spring-service-beans-dao.xml`

```xml
<bean id="incidentServiceTarget"
      class="org.fusesource.devoxx.reportincident.service.impl.IncidentServiceImpl">
  <property name="incidentDAO">
    <osgi:reference interface="org.fusesource.devoxx.reportincident.dao.IncidentDAO"/>
  </property>
</bean>

<bean id="abstractService" abstract="true"
      class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
  <property name="transactionManager">
    <osgi:reference id="txManager" filter="(tx=JPA)"
              interface="org.springframework.transaction.PlatformTransactionManager"/>
  </property>
</bean>

<bean id="incidentService"
      parent="abstractService"
      class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

  <property name="target">
    <ref bean="incidentServiceTarget" />
  </property>

  <property name="proxyInterfaces">
    <list>
      <value>org.fusesource.devoxx.reportincident.service.IncidentService</value>
    </list>
  </property>

  <property name="transactionAttributes">
    <props>
      <prop key="*">PROPAGATION_REQUIRED</prop>
    </props>
  </property>
</bean>
```

STEP 4 : The last step requires that we expose our Service on the OSGI container to allow the Apache Camel route as well as the Apache Wicket web application to use it

```xml
<!-- Expose Service as OSGI -->
<osgi:service ref="incidentService"
              interface="org.fusesource.devoxx.reportincident.service.IncidentService"/>
```

After the creation of the Service layer, we have finished to put in place the DAO / Services layers, Transaction Management and register the Services for the CRUD. It is time now to design the web service, create the camel routes and install the Web project

WEB SERVICES

To generate the code that we need to expose the WebService from the Apache Camel route, we will simply register the Apache CXF Maven plugin in the pom.xml file and place the contract (= wsdl) file under the directory src/main/resources/META-INF of webservice mavem module (already done)

STEP 1 : Check what we have added in the `pom.xml` file to generate the code from the wsdl

```xml
<plugin>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-codegen-plugin</artifactId>
  <version>${cxf-version}</version>
  <executions>
    <execution>
      <id>generate-sources</id>
      <phase>generate-sources</phase>
      <configuration>
        <sourceRoot>${basedir}/target/generated/src/main/java</sourceRoot>
        <wsdlOptions>
          <wsdlOption>
            <wsdl>${basedir}/src/main/resources/META-INF/wsdl/report_incident.wsdl</wsdl>
          </wsdlOption>
        </wsdlOptions>
      </configuration>
      <goals>
        <goal>wsdl2java</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

STEP 2 : The code generated contains JAXB model classes (`InputReportIncident` and `OutputReportIncident`), Interface using @WebService annotations to expose the Web Service and a java class which a factory to access the class

Interface

```java
@WebService(targetNamespace = "http://reportincident.devoxx.fusesource.org",
            name = "ReportIncidentEndpoint")
@XmlSeeAlso({ObjectFactory.class})
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
public interface ReportIncidentEndpoint {

    @WebResult(name = "outputReportIncident",
               targetNamespace = "http://reportincident.devoxx.fusesource.org",
               partName = "out")
    @WebMethod(operationName = "ReportIncident",
               action = "http://reportincident.devoxx.fusesource.org/ReportIncident")
    public OutputReportIncident reportIncident(
        @WebParam(partName = "in", name = "inputReportIncident",
                  targetNamespace = "http://reportincident.devoxx.fusesource.org")
        InputReportIncident in
    );

}
```

Service

```java
@WebServiceClient(name = "ReportIncidentEndpointService",
                  wsdlLocation = "file:full/path/to/report_incident.wsdl",
                  targetNamespace = "http://reportincident.devoxx.fusesource.org")
public class ReportIncidentEndpointService extends Service {

    public final static URL WSDL_LOCATION;

    public final static QName SERVICE =
            new QName("http://reportincident.devoxx.fusesource.org",
                      "ReportIncidentEndpointService");
    public final static QName ReportIncidentPort =
            new QName("http://reportincident.devoxx.fusesource.org", "ReportIncidentPort");

    public ReportIncidentEndpointService(URL wsdlLocation) {
        super(wsdlLocation, SERVICE);
    }

    public ReportIncidentEndpointService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public ReportIncidentEndpointService() {
        super(WSDL_LOCATION, SERVICE);
    }


    /**
     *
     * @return
     *     returns ReportIncidentEndpoint
     */
    @WebEndpoint(name = "ReportIncidentPort")
    public ReportIncidentEndpoint getReportIncidentPort() {
        return super.getPort(ReportIncidentPort, ReportIncidentEndpoint.class);
    }

    /**
     *
     * @param features
     *     A list of {@link javax.xml.ws.WebServiceFeature} to configure on the proxy.
     *     Supported features not in the <code>features</code> parameter will have their
     *     default values.
     * @return
     *     returns ReportIncidentEndpoint
     */
    @WebEndpoint(name = "ReportIncidentPort")
    public ReportIncidentEndpoint getReportIncidentPort(WebServiceFeature... features) {
        return super.getPort(ReportIncidentPort, ReportIncidentEndpoint.class, features);
    }

}
```

ROUTING

The routing/mediation between "services" will be created using Camel Spring DSL language. We will describe its creation/genesis step by step

- Define ActiveMQ component

- Define Beans that Apache Camel will use to insert records in DB (IncidentSaver), extract info from the webservice (WebService.java) and prepare the response for the Web Service (Feedback.java)

- Configure CXF endpoint

- Camel routes and DataFormats

STEP 1 : Edit the file `src/main/resources/META-INF/spring`

/camel-context.xml (in the routing module) and add the following bean for the ActiveMQ component

```xml
<!-- ActiveMQ component -->
<bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="connectionFactory">
    <osgi:reference interface="javax.jms.ConnectionFactory"/>
  </property>
  <property name="transacted" value="true"/>
  <property name="transactionManager">
    <osgi:reference
        interface="org.springframework.transaction.PlatformTransactionManager"/>
  </property>
</bean>
```

Remarks :

We retrieve the connectionFactory (aka Datasource, connectionPool for DB) using OSGI Service

The component is transactional and can partipicipate in a Transaction. Session will be transacted with the broker

We use the Fuse ESB Transaction Manager exposed under the interface

`org.springframework.transaction.PlatformTransactionManager`

STEP 2 : Add beans definition

```xml
<bean id="incidentSaver"
      class="org.fusesource.devoxx.reportincident.internal.IncidentSaver">
  <property name="incidentService">
    <osgi:reference
        interface="org.fusesource.devoxx.reportincident.service.IncidentService"/>
  </property>
</bean>

<bean id="webservice" class="org.fusesource.devoxx.reportincident.internal.WebService"/>

<bean id="feedback" class="org.fusesource.devoxx.reportincident.internal.Feedback"/>
```

Remark : The `IncidentSaver` class has a property IncidentService and we will inject its beab proxy using Spring

STEP 3 : Configure the CXF endpoint

```xml
<import resource="classpath:META-INF/cxf/cxf.xml"/>
<!-- webservice endpoint -->
<cxf:cxfEndpoint id="reportIncident"
    address="http://localhost:8282/cxf/camel-example/incident"
    serviceClass="org.fusesource.devoxx.reportincident.ReportIncidentEndpoint"
    xmlns:s="http://reportincident.devoxx.fusesource.org">
</cxf:cxfEndpoint>
```

Remark :
CXF uses Jetty (Web Applications Server project managed by Eclipse community) to expose the Web Service. In this case, we will create a web server instance running on the port 8282.

The webservice uses the `ReportIncidentEndpoint` interface created in the project webservice. This is this service that will receive the SAOP envelope, extract the body. When the Apache Camel route will generate the reponses, then it will prepare the SOAP message for the HTTP client

STEP 4 : Define the transaction policy that Apache Camel will use

```xml
<!-- Tx Manager -->
<osgi:reference id="txManager" filter="(tx=JPA)"
                interface="org.springframework.transaction.PlatformTransactionManager"/>

<!-- PROPAGATION used for Transactions-->
<bean id="PROPAGATION_REQUIRED"
      class="org.apache.camel.spring.spi.SpringTransactionPolicy">
  <property name="transactionManager" ref="txManager"/>
  <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/>
</bean>
```

Remark : we will reuse the JPA Transaction manager created in the persistence module and exposed as an OSGI service

STEP 5 : Create the Camel Context

```xml
<camelContext trace="true" xmlns="http://camel.apache.org/schema/spring">
</camelContext>
```

STEP 6 : Add the data format definition that Apache Camel will use to transform CSV record into objects (or objects into CSV)

```xml
<camelContext trace="true" xmlns="http://camel.apache.org/schema/spring">
 <dataFormats>
   <bindy type="Csv" id="bindyDataFormat"
          packages="org.fusesource.devoxx.reportincident.model"/>
  </dataFormats>
</camelContext>
```

STEP 7 : Create the route which will poll files from a directory, transform the content into a
ReportIncident and place it into a queue

```xml
<route id="fileToQueueIn">
  <from uri="file://data/reportincident/?move=done/${date:now:yyyyMMdd}/${file:name}.bak"/>
  <setHeader headerName="origin">
    <constant>file</constant>
  </setHeader>
  <unmarshal ref="bindyDataFormat"/>
  <to uri="activemq:queue:in"/>
</route>
```

STEP 8 : Expose a webservice, send the response to the caller and place ReportIncident object into the
queue

```xml
<route id="webServiceToQueueIn">
  <from uri="cxf:bean:reportIncident"/>
  <setHeader headerName="origin">
    <constant>webservice</constant>
  </setHeader>
  <convertBodyTo type="org.fusesource.devoxx.reportincident.InputReportIncident"/>
  <to uri="bean:webservice"/>
  <inOnly uri="activemq:queue:in"/>
  <transform>
    <method bean="feedback" method="setOk"/>
  </transform>
</route>
```

STEP 9 Create the camel transactional route which will insert records into the h2 DB

```xml
<route id="queueIntoIncidentSaver">
  <from uri="activemq:queue:in"/>
  <transacted ref="PROPAGATION_REQUIRED"/>
  <to uri="bean:incidentSaver?method=process"/>
</route>
```

STEP 10 : Create the following classes in the directory `src/main/java/org/fusesource/devoxx/`
`reportingincident/internal`

WebService

```java
package org.fusesource.devoxx.reportincident.internal;

/* SNIP */

public class WebService {

    public void process(Exchange exchange) throws ParseException {

        InputReportIncident webincident = (InputReportIncident)exchange.getIn().getBody();

        List<Map<String, Incident>> models = new ArrayList<Map<String, Incident>>();
        Map<String, Incident> model = new HashMap<String, Incident>();

        // Convert the InputReportIncident into
        // an org.apache.camel.example.reportincident.model.Incident instance
        Incident incident = new Incident();

        DateFormat format = new SimpleDateFormat( "dd-MM-yyyy" );
        incident.setIncidentDate(format.parse(webincident.getIncidentDate()));

        incident.setDetails(webincident.getDetails());
        incident.setEmail(webincident.getEmail());
        incident.setFamilyName(webincident.getFamilyName());
        incident.setGivenName(webincident.getGivenName());
        incident.setIncidentRef(webincident.getIncidentId());
        incident.setPhone(webincident.getPhone());
        incident.setSummary(webincident.getSummary());

        // Get Header origin from message
        String origin = (String) exchange.getIn().getHeader("origin");

        // Specify current Date
        format = new SimpleDateFormat( "dd/MM/yyyy HH:mm:ss" );
        String currentDate = format.format( new Date() );
        Date creationDate = format.parse( currentDate );

        incident.setCreationDate(creationDate);
        incident.setCreationUser(origin);

        // and place it in a model (cfr camel-bindy)
        model.put(Incident.class.getName(), incident);
        models.add(model);

        // replace with our input
        exchange.getOut().setBody(models);

        // propagate the header
        exchange.getOut().setHeader("origin", origin);

    }
}
```

IncidentSaver

```java
package org.fusesource.devoxx.reportincident.internal;

import org.apache.camel.Exchange;
import org.fusesource.devoxx.reportincident.model.Incident;
import org.fusesource.devoxx.reportincident.service.IncidentService;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

public class IncidentSaver {

    private IncidentService incidentService = null;

    public void process(Exchange exchange) throws ParseException {

        List<Map<String, Object>> models = new ArrayList<Map<String, Object>>();
        Map<String, Object> model = new HashMap<String, Object>();

        // Get models from message
        models = (List<Map<String, Object>>) exchange.getIn().getBody();

        // Get Header origin from message
        String origin = (String) exchange.getIn().getHeader("origin");

        Iterator<Map<String, Object>> it = models.iterator();

        // Specify current Date
        DateFormat format = new SimpleDateFormat( "dd/MM/yyyy HH:mm:ss" );
        String currentDate = format.format( new Date() );
        Date creationDate = format.parse( currentDate );

        while (it.hasNext()) {

            model = it.next();

            for (String key : model.keySet()) {

                // Retrieve incident from model
                Incident incident = (Incident) model.get(key);
                incident.setCreationDate(creationDate);
                incident.setCreationUser(origin);

                // Save org.fusesource.devoxx.reportincident.model.Incident
                incidentService.saveIncident(incident);
            }

        }

    }

    // Property used to inject service implementation
    public void setIncidentService(IncidentService incidentService) {
        this.incidentService = incidentService;
    }
}
```

Feedback

```java
package org.fusesource.devoxx.reportincident.internal;

import org.fusesource.devoxx.reportincident.OutputReportIncident;

public class Feedback {

    public OutputReportIncident setOk() {
        OutputReportIncident outputReportIncident = new OutputReportIncident();
        outputReportIncident.setCode("0");
        return outputReportIncident;
    }

}
```

```java
package org.fusesource.devoxx.reportincident.internal;

import org.fusesource.devoxx.reportincident.OutputReportIncident;
```

BUILDING AND RUNNING

Build the code

At the route of the project, execute the command

```
mvn clean install
```

Setup the Database

STEP 1 : Open a DOS/UNIX console in the folder persistence/database

STEP 2 : Download H2 Database (http://www.h2database.com/html/download.html) and install it

STEP 3 : Start H2 Server using the bat or shell script

```
./h2.sh &
```

The H2 server is started and to manage the databases from your web browser, simply click on the following url http://localhost:8082/

STEP 4 : Next create the report database

In the login.jsp screen, select Generic (H2) - Server
Add as settings name : Generic H2 (Server) - Webinar
and modify the JDBC url as such : `jdbc:h2:tcp://localhost/~/reportdb`

Next click on "connect" and the screen to manage the reportdb appears

STEP 5 : Create Schema and Tables using the script located in the file `db/src/main/config/h2-script.sql`

Execute the scripts 1), 2) and 3) defined in this file

Check that the records are well created using the command : `SELECT * FROM REPORT.T_INCIDENT;`

Deploy and test

STEP 1 : Unzip `apache-servicemix-4.4.1-fuse-01-06.zip` or tar.gz in the directory `~/devoxx/servers/` or `D:\devoxx\servers`

STEP 2 : In a DOS/Unix console, change to the directory `apache-servicemix-4.4.1-fuse-01-06/bin`

STEP 3 : Edit the file `apache-servicemix-4.4.1-fuse-01-06/etc/org.apache.karaf.features.cfg` and replace the featuresBoot line by this one :

```
featuresBoot=karaf-framework,config,war,activemq-broker,activemq-spring,camel,camel-
cxf,camel-activemq,camel-jaxb,camel-bindy,jpa
```

STEP 5 : Start ServiceMix `servicemix.bat` or `./servicemix`

STEP 6 : In the servicemix console, launch the commands to install features of reportincident demo

```
features:addUrl mvn:org.fusesource.devoxx.reportincident/features/1.0-SNAPSHOT/xml/features
features:install reportincident-jpa
```

STEP 7 : Check if the project works fine and connect to the following url in your browser

http://localhost:8282/cxf/camel-example/incident?wsdl

STEP 8 : Verify that you have records in the table screen

http://localhost:8181/reportincidentweb/

STEP 9 : Copy incident file from the directory `~/devoxx/` or `D:\devoxx`

```
cp ~/devoxx/routing/src/data/csv-one-record.txt ~/devoxx/servers/apache-
servicemix-4.4.1-fuse-01-06/data/reportincident/
```

STEP 10 : Use SOAPUi client and send this envelope to the server

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:rep="http://reportincident.devoxx.fusesource.org">
  <soapenv:Header/>
  <soapenv:Body>
    <rep:inputReportIncident>
       <incidentId>999</incidentId>
       <incidentDate>10-05-2011</incidentDate>
       <givenName>Charles</givenName>
       <familyName>Moulliard</familyName>
       <summary>Issue at the DevoXX</summary>
       <details>Room is burning !</details>
       <email>cmoulliard@fusesource.com</email>
       <phone>+32473604014</phone>
    </rep:inputReportIncident>
  </soapenv:Body>
</soapenv:Envelope>
```

THANKS FOR ATTENDING!