Hands on with Apache ActiveMQ, ServiceMix, Camel and CXF

Devoxx 2011

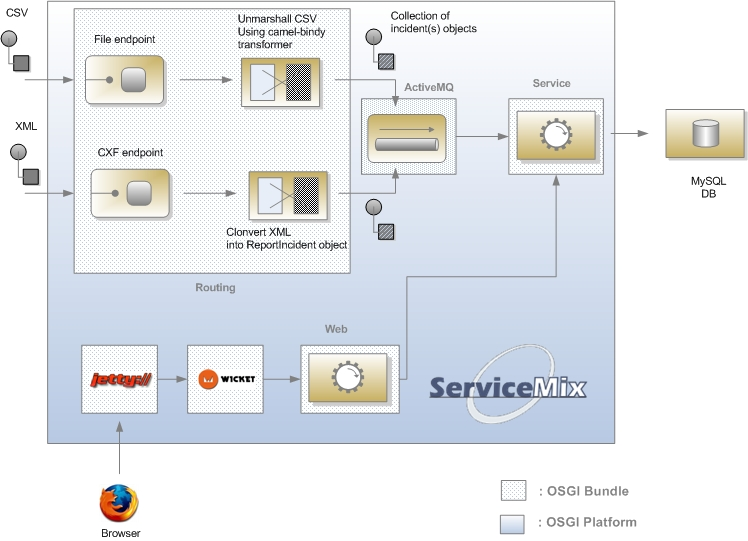
Charles Moulliard and Gert Vanthienen

**Table Of Contents**

Word did not find any entries for your table of contents.  
In your document, select the words to include in the table of contents, and then in the Formatting Palette under Styles, click a heading style. Repeat for each heading that you want to include, and then insert the table of contents in your document. You can also create a table of contents by clicking the Create with Manual Formatting option and then type the entries manually.

Introduction

The goal of this Devoxx “hand 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 after



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 MANIFEST.mf file has been modified to include the OSGI METADATA (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 simply 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 D:/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

Develop 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 : 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

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);

}

}

**Incident class**

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 String givenName;

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 annotation

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 to map the CSV record field with the java properties using Jabva Annotations as we do with JAXB

Add the following annotations in the Incident class

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;

Extend model layer

Using JPA, our model class will become an Entity and will be used next to insert records in the database

Our class will be mapped with T\_INCIDENT table

Definition of the different columns, their name and filed length is provided here after

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

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;.

STEP 2 : Define the persistence file in the directory of the persistence-jpa maven module /src/main/resources/META-INF/persistence.xml

STEP 3 : Add the definition to map Entity with its provider

<?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

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

STEP 1 : Create the file persistence-dao file under src/main/resources/META-INF/spring

STEP 2 : Add

<!--

Activates various annotations to be detected in bean classes: Spring's

@Required and @Autowired, as well as JSR 250's @PostConstruct,

@PreDestroy and @Resource (if available) and JPA's @PersistenceContext

and @PersistenceUnit (if available).

-->

<context:annotation-config/>

<!-- enables interpretation of the @PersistenceUnit/@PersistenceContext annotations providing convenient

access to EntityManagerFactory/EntityManager -->

<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>

<!-- 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"/>

<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>

<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>

<bean id="dataSourceH2" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<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>

<!-- TransactionManager is required -->

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory"/>

<property name="dataSource" ref="dataSourceH2"/>

</bean>

<!-- 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 3 : Create the DAO Interface IncidentDAO

package org.fusesource.devoxx.reportincident.dao;

import java.util.List;

import org.fusesource.devoxx.reportincident.model.Incident;

public interface IncidentDAO

{

public abstract Incident getIncident(long paramLong);

public abstract List<Incident> findIncident();

public abstract List<Incident> findIncident(String paramString);

public abstract void saveIncident(Incident paramIncident);

public abstract void removeIncident(long paramLong);

}

STEP 4 : Its implementation IncidentDAOImpl

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();

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_