Design of the OSS Common API Reference Implementation (JSR 144)

OSS through Java™ Initiative

Vincent Perrot, Sun Microsystems Inc.

COM-API-Ri_Design.1.3.1.doc

Executive Summary

The Common API offers interfaces and classes, which are common across all OSS API defined under OSS J Initiative. This document describes how the Reference Implementation (RI) was created and designed to implement the Common API Specifications. The RI contains several parts that could be used independently:

- the based interfaces implementation
- the CBE implementation
- examples of java, xml over JMS and Web services implementation.

Table of Contents

Preface

Objectives

Design description of the OSS/J Common Reference Implementation

Audience

The target audiences are

- Developers who seek information about how the Common API can be implemented
- Developers of other OSS/J API Reference Implementers
- Developers who want to make use of these API and extend its implementations

Related Information

Prerequisite

« Java EE Tools Bundle Beta », containing JSE 5, JEE, netbeans 5.5, and OpenESB. You can download it at:

http://java.sun.com/javaee/downloads/index.jsp

Revision History:

Date	Version	Author	State	Comments
February 2006	1.3	Vincent Perrot Sun Microsystems Inc	Maintenance release 3	
September	1.4	Vincent Perrot Sun Microsystems	Maintenance	Change all the content

2006		Inc	Release 4	according to the new RI design

1 Introduction

This document describes the design and the creation phases of the OSS through Java™ Initiative, Common API Reference Implementation.

The Reference Implementation can be used either as a proof-of-concept for the Common API specification, showing that it is possible to implement the API or API's can be directly used as a package. The Reference Implementation consists of two parts. The first part, consists of abstract classes and interfaces which can be reused by the target audiences of this API directly and the second part consists of concrete implementation the Interfaces of the API this part cant be reused and this only serves as proof or example how to implement the API.

This document shows how the Reference Implementation is designed. Reference Implementation provides the concrete interfaces and classes as specified in the Common API Specifications. It also provides some abstract classes for certain interfaces so that the developers classes can directly extend these classes and implement only those none generic methods. Finally it contains examples of the different profile implementations.

In general the concrete Reference Implementation is designed as a set of Enterprise Java Beans. The entire RI is developed using netbeans and J2E SDK.

2 Design Overview of Reusable Part of Reference Implementation

The Reference Implementation has following set of classes apart from the classes and interfaces specified by the specification. Each of these classes is explained in detail in later part of the document.

- Application Context Implementer class
- Attribute Access classes
- Base Java Value Type and CBE implementations

All the reusable classes are archived in the jars with a name starting by oss_cbe for the CBE implementations and oss_common_ri for the implementation of the based interfaces.

In the following chapter some interface implementations are detailed.

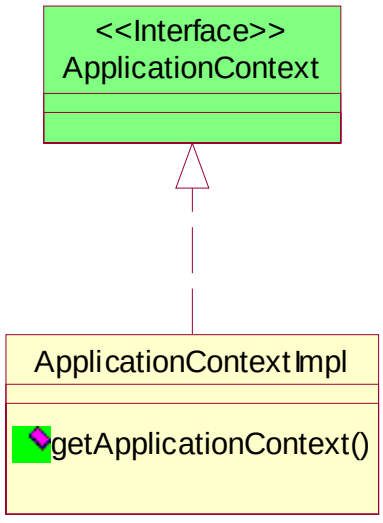
2.1 Application Context Implementer class

The ApplicationContext implementation class contains the URL and other system properties required to set up an initial connection with the JNDI provider into which the components in charge of that managed entity are registered. This class implements the ApplicationContext Interface defined in the specification apart from this it provides additional static method, which provides the Application context based on the present server configuration.

The figure below shows the relationship between the interface and class

Green color indicates the Interface is part of the specification

Yellow class is additional class defined in Reference Implementation



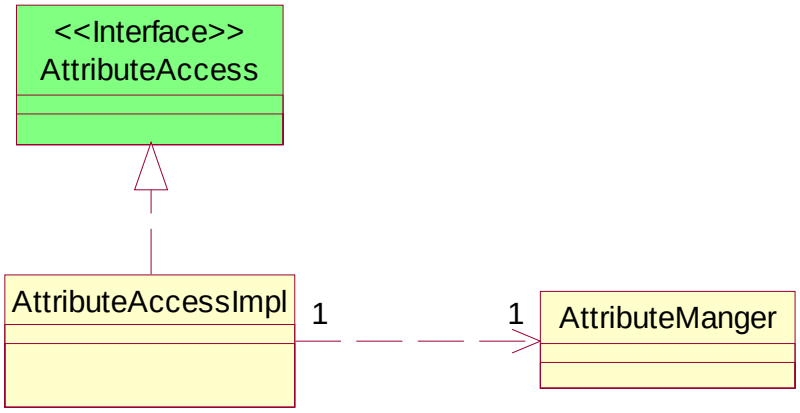
2.2 Attribute Access Classes

Two classes namely AttributeManager and AttributeAccessImpl are the classes, which help in accessing the attributes of value object as specified in the specification. Attribute Manager manages the attributes of the value object and provides easy methods to get the properties of attributes like attribute names, settable attributes etc. AttributeAccessImpl is abstract class implementing the AttributeAcess Interface, which all value objects must implement according to the specification. This abstract class manages the attributes using the AttributeManager class.

The figure below shows the relationship between the interface and classes

Green color indicates the Interface is part of the specification

Yellow classes are addition classes defined in Reference Implementation



2.3 Base Java Value Type and CBE implementations

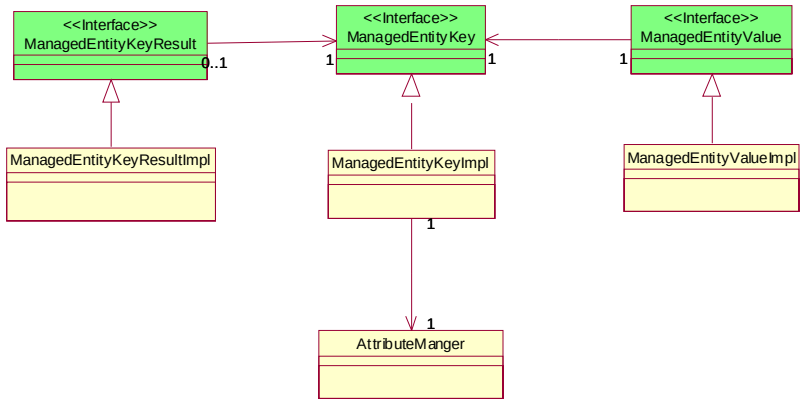
The Java Value Types are the objects, which are exchanged between the client and JVTSessionBean.The following classes are defined in addition to those specified in the specification

- ManagedEntityKeyImpl
- ManagedEntityKeyResultImpl
- ManagedEntityValueImpl

The figure below shows the relationship between the interface and classes

Green color indicates the Interface is part of the specification

Yellow classes are addition classes defined in Reference Implementation



ManagedEntityKeyImpl is an abstract class, which implements the ManagedEntityKey Interface. Since very managed Entity type must have a ManagedEntityKey Interface implementation it can directly extend ManagedEntityKey abstract class and define the methods to make the primary key which will be specific to the entity type, functionalities like checking equality of two key objects as specified in the specification is taken care by the abstract class. ManagedEntityvalueImpl implements the ManagedEntityValue Interface as specified in the specification and it also extends the AttributeAccessImpl class.

Then all the CBE implementation classes are derived from these based definitions.

The CBE implementation classes have been generated for a large part using a hand made java generator. The generation facilitates:

- the consistencies in names and formats
- the fastidious duplication of “common” code section (managing inheritance and Attribute management, population etc)
- stable clone and equality methods generation
- etc.

3 Design overview of integration profile examples

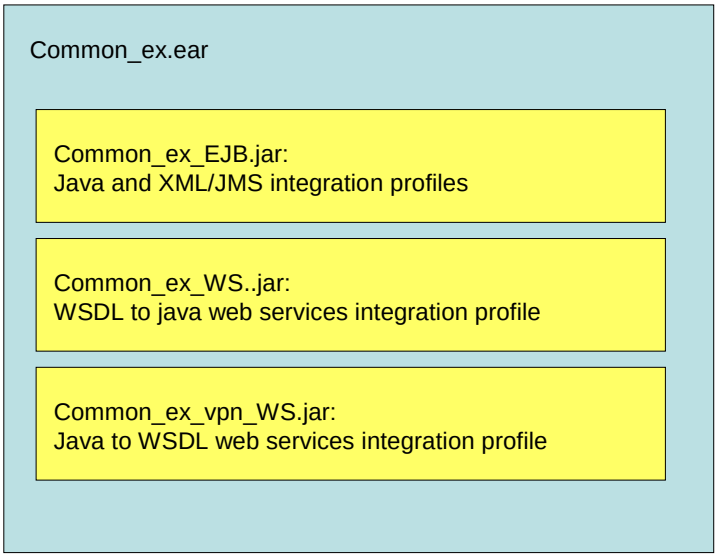
This part of Reference Implementation consists of concrete example Implementations of API for the three integration profiles with a special attention for the web services one.

The example application (a minimalist vpn services) and its three integration profiles are packaged and deployed within a unique enterprise archive (ear file). It provides any client application all the possible interfaces to the “common” features. And it demonstrates the extension capabilities of the Common API design.

Netbeans and Java EE SDK have been largely used for this development. Java EE 5 SDK supports the combination of J2EE 1.4 EJBs and latest Web services from Java EE 5 into the same archive. This capability has been used to build a simple and reusable OSSJ example.

For the implementation of the web services interfaces 2 different approaches have been used:

- WSDL to java: generation of the WS implementation from the WSDL. This technique have been used to produce the “common” features based on the WSDL generated in the Specification bundle.
- Java to WSDL: the code of the specification have been duplicated into the RI and then instrumented using JSE5 annotation in order to code the Webservices endpoint and then generate the WSDL. This technique have use for the vpn example.

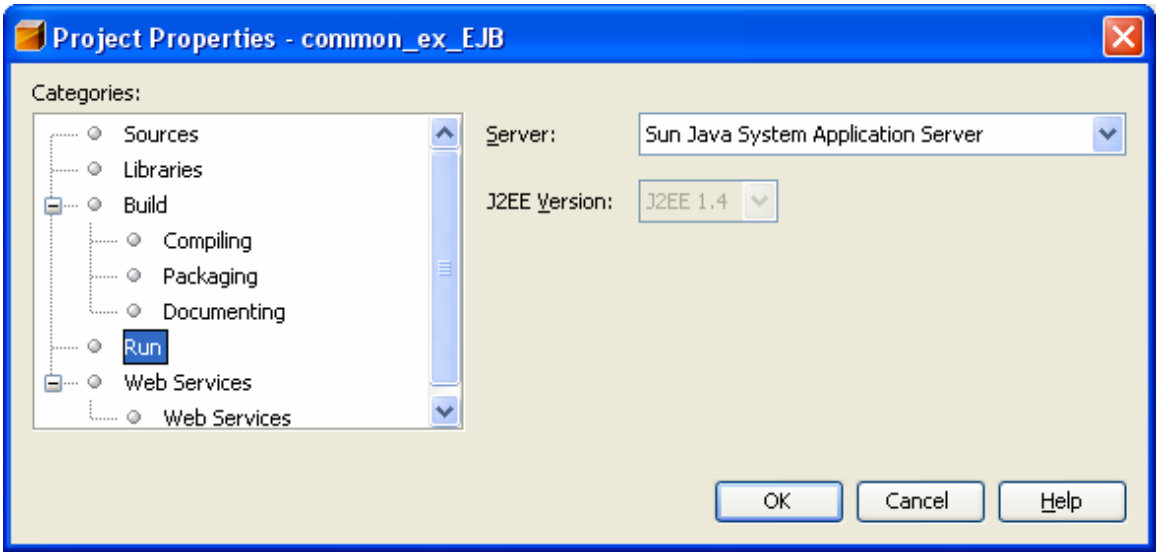


3.1 Common_ex_EJB

This module contains all the code necessary to implement the applicaition itself (vpn service) and the java and XML/JMS integration profiles example. It uses JSE5 (including annotation that will be used later only for the WS implementation). It includes :

- the specification interfaces (javax.oss package name),
- the class implementations of the used CBE (ossj.common pacjage name) as well as
- the EJBs and necessary message driven bean needed for the XML/JMS profiles.

This module is not using any Java EE 5 specific feature and reference only J2EE 1.4 specification.



The EJB Module common_ex_EJB is created as a new project where the necessary source files have been either newly created or copying form other projects.

This module has no external dependencies (libraries or other projects) except to Java SE and EE.

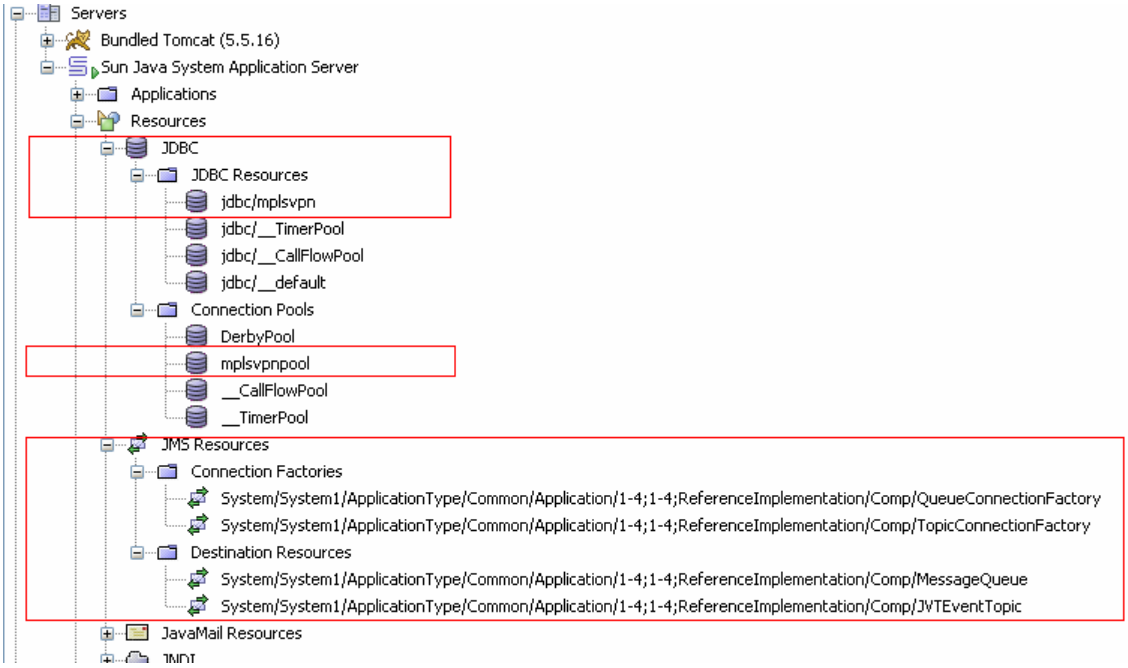
The deployment descriptor follows the OSSJ convention for JNDI names, etc.

It contains four EJBs implementing the Vpn applicaition:

- VpnServiceSB: this is the Session bean (java integration profile) exposing the Remote interface of the application and extending the Common interface.

- JmsSender and XVTMessageDrivenBean handling the XML resquests and responses of the XML/JMS profile
- MplsvpnTblEB, the Entity bean implementing the application core business and persistency following the recommended OSSJ design guidelines.

This application also needs the following external resources to deploy and run correctly:



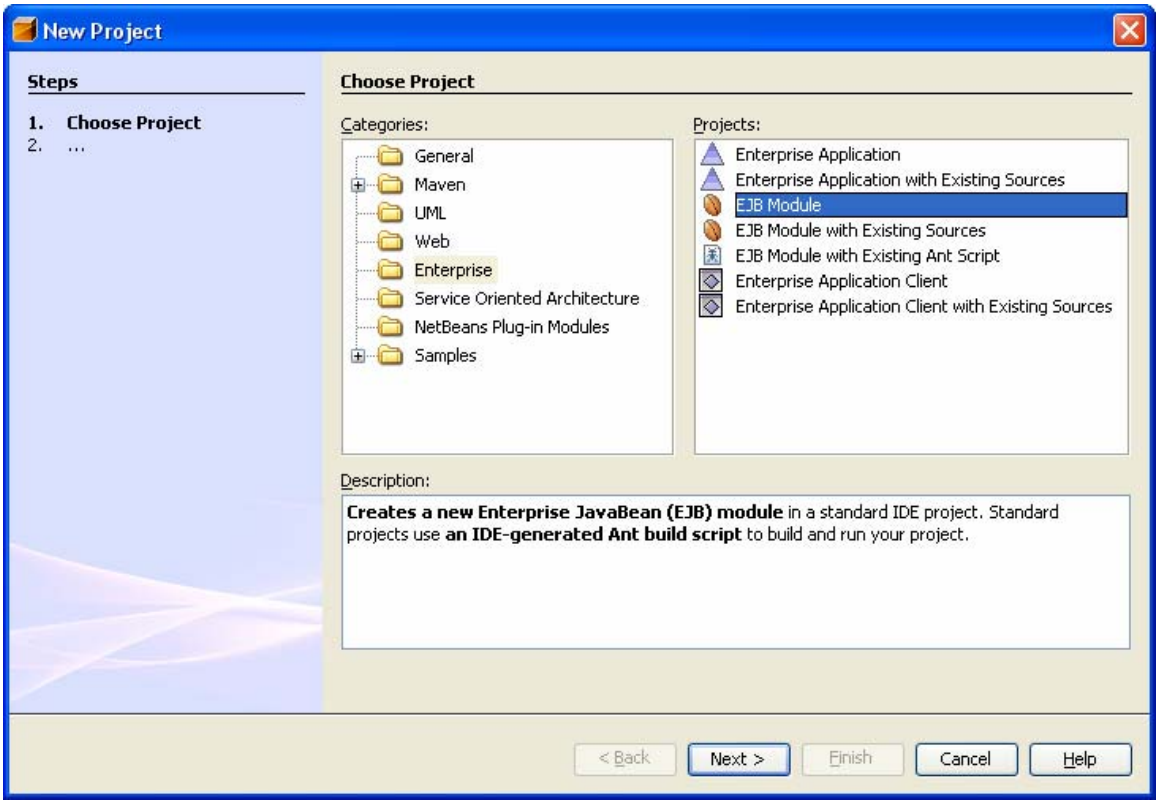
Once the started the database table need to be created using the following SQL syntax (the src directory sql/create.sql file):

```
-- CONNECT 'jdbc:derby://localhost:1527/mplsvpndb;user=dbuser;password=dbpassword;';

CREATE TABLE MPLSVPN_TBL (
  MplsKey varchar(128) not null,
  State varchar(128),
  SubscriberId varchar(256),
  Mandatory int,
  VrfName varchar(256),
  StartMode int,
  CONSTRAINT mplsvpntable_PK Primary Key (MplsKey));

-- EXIT;
```

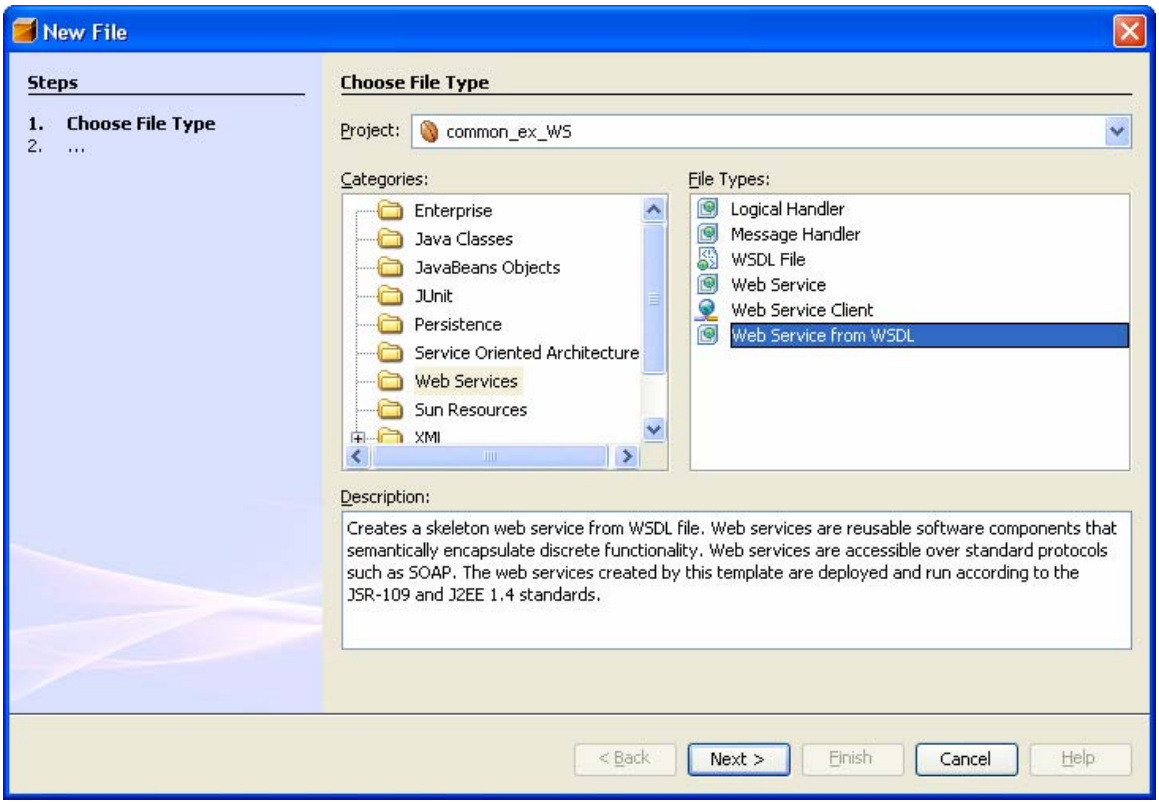
3.2 Common_ex_WS

This module contains the WS profile for the common services. It has been generated using Netbeans capabilities from the WSDL files.

After the creation of the project (EJB Module) in netbeans,



create the WS using the “new file” wizard:



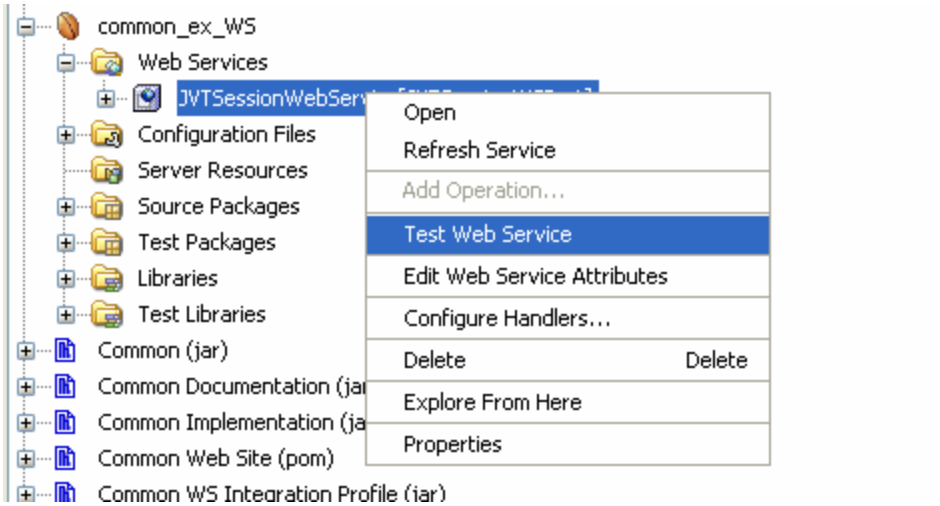
Then follow instructions, and finally complete the code skeleton.

The following lines needs to be added to interface the session bean implementing the VPN service:

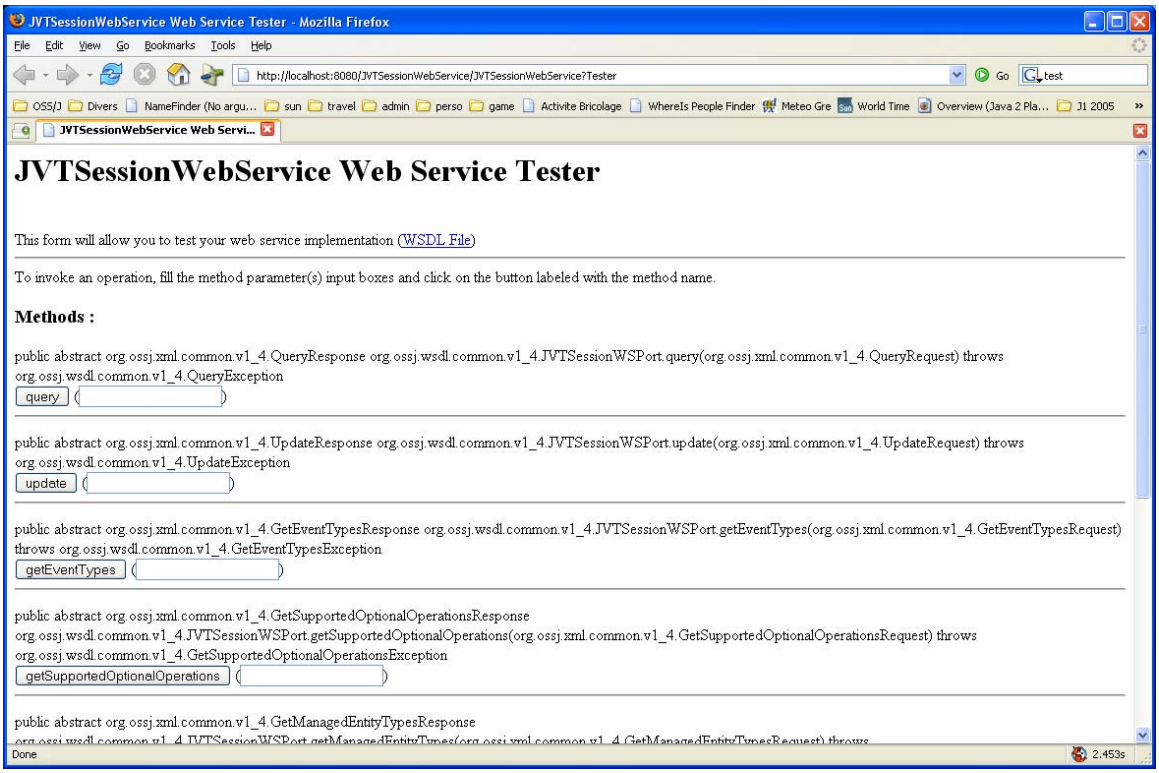
```
    @EJB(name="VpnServiceEJB")
    VpnServiceRemoteHome vpnHome;
…
    private VpnServiceRemote getSession() throws javax.ejb.CreateException,
javax.naming.NamingException {
        if (initCtx == null) initCtx = new InitialContext();
        vpnHome =
(VpnServiceRemoteHome)initCtx.lookup("System/System1/ApplicationType/Common/Applicat
ion/1-4;1-4;ReferenceImplementation/Comp/VpnServiceBean");
        if (vpnHome == null){
            throw new javax.ejb.CreateException("Lookup of VpnServiceRemoteHome
failed");
        }
        try {
            return vpnHome.create();
        } catch (java.rmi.RemoteException rex) {
            rex.printStackTrace();
            throw new EJBException("RemoteException: " + rex.getMessage());
        }
    }
```

Once deployed, this service could be easily test using netbeans and JEE 5 SDK capabilities:

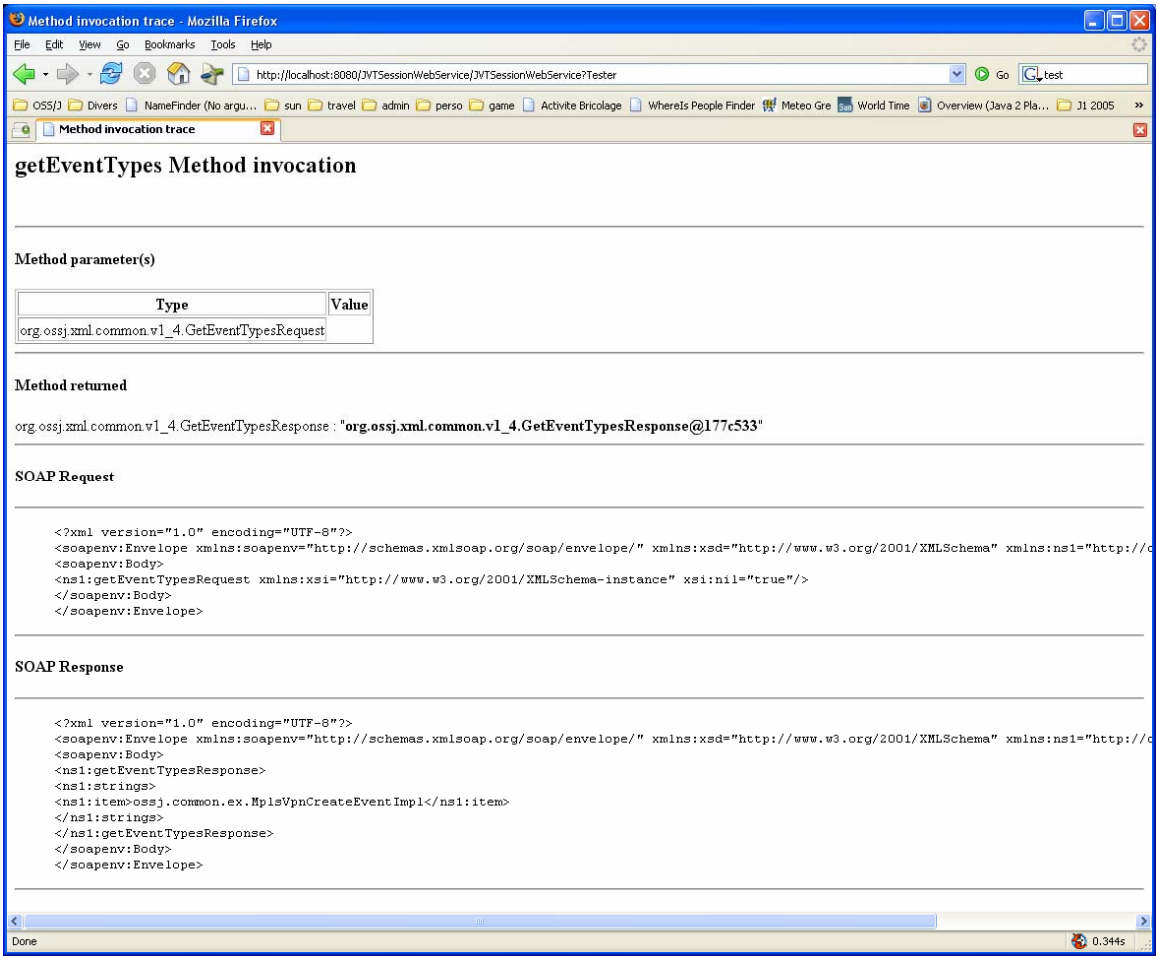
Go to the common_ex_WS project, open the “Web Services” section right – click on the service and selection “Test Web Service”:



Your favorite web browser will pops up the web services page from where you can exercise one of the OSS Common API feature:



For example, pushing the “getEventTypes” button will return:



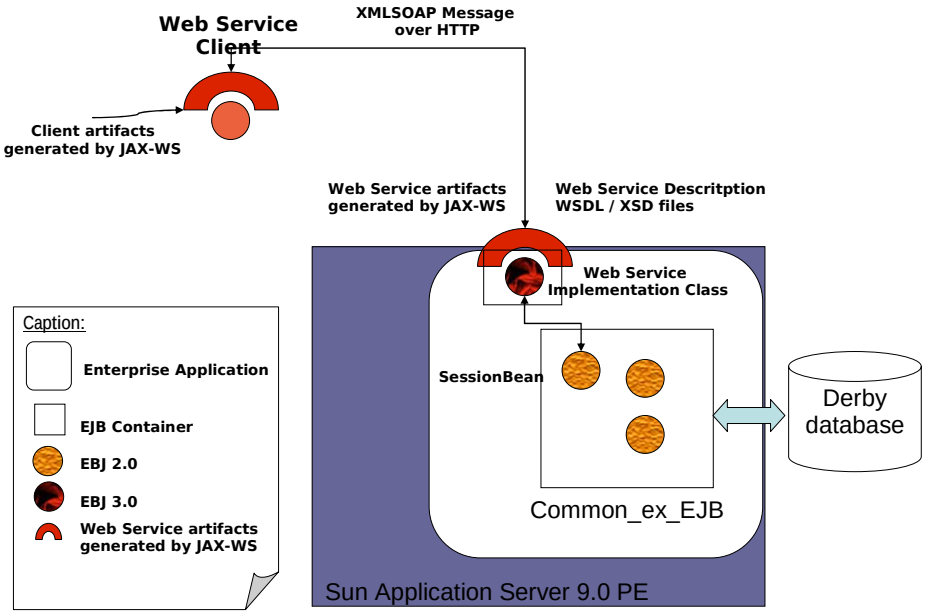
3.3 Common_ex_vpn_WS

The module demonstrates how to expose a JEE enterprise application – the java integration profile of module common_ex_EJB- as web service with JAX-WS 2.0. Concretely, a new EJB3.0 implementing the web service have been created. This EJB3.0 accesses the existing Session Bean to invoke the corresponding methods providing the services. The obsolete JAX-RPC tools, have been replaced by JAX-WS 2.0 tools to generate the artifacts exposing the EJB3.0 as web service. In the following sections, the migration process is detailed step by step, with comments to better understand constraints and choices. This section comes with a netbeans project illustrating the work with a concrete example.

This section is divided into four parts:

- the work environment and project creation,
- the generation of the web service endpoint,

- the packaging of the project deployment file (the EAR),
- the test of the web services.



3.3.1 work environment

« Java EE Tools Bundle Beta », containing JSE 5, JEE, netbeans 5.5, and OpenESB. You can download it at:

http://java.sun.com/javaee/downloads/index.jsp

JAX-WS enables to generate the web service from a Java Bean. This previous edition of Sun Application Server includes wsgen and wsimport, the two JAX-WS tools used here.

3.3.2 generation of the web service endpoint

It starts from the creation of a new EJB Module named common_ex_vpn_WS and a new Web Service named VpnService (More details will be given later).

This new module is naturally included into the final ear as follow:

In case the enterprise application module have not yet been created:

- create a new enterprise application, choose JEE 5;
- do not create any related module : no EJB module, no web application module, no client application module;

then link this EJB module to it; and finally clean and build your project (from the EAR ant targets, more details are given later).

The module is now ready to use JAX WS tools. The Web services end point will generated from the annotated EJB, interfaces and classes. This will show how existing systems can expose web services integration profile. (Note: a minimalist approach is applied. Many other customizations could be made to make the final generated WSDL files aligned with OSSJ standard).

There are two possible implementation: either to interface the session bean or the underlying entity bean (bean called by the session bean):

- first solution would be to create a new EJB 3.0 for the web service, exposing the VPN service interface. This Ejb 3.0 will then communicate with the Session Bean 2.0 of 'common_ex_EJB' module.
- Second solution would be to convert the existing Session Bean in the web service implementation (in annotated EJB 3.0) so that the web service implementation class access directly the entity bean.

To minimize the impact on the existing Session Bean, and to preserve the Java integration profile, the first solution has preferred.

In this netbean project create a new EJB module where the desired methods are created using the same names as the Java integration profile. Then it is “connected” to the Session Bean exposing the java integration profile as follow:

```
/*
 * VpnService.java
 *
```

```
 * Created on July 7, 2006, 3:40 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package ossj.common.ex;

import com.infomodel.MplsVpnServiceKey;
import com.infomodel.MplsVpnServiceValue;
import java.rmi.RemoteException;
import javax.ejb.EJB;
import javax.ejb.EJBException;
import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.naming.InitialContext;
import ossj.common.AttributeAccessImpl;

/**
 *
 * @author vince
 */

@Stateless()
@WebService(wsdlLocation="META-INF/wsdl/VpnServiceService.wsdl")
public class VpnService {

    @EJB(name="VpnServiceEJB")
    VpnServiceRemoteHome vpnHome;

    InitialContext initCtx = null;



    @WebMethod(operationName="createMplsVpnServiceByValue")
    public MplsVpnServiceKey  createMplsVpnServiceByValue(MplsVpnServiceValue
mplsVpnServiceValue)
    throws javax.oss.OssIllegalArgumentException,
            java.rmi.RemoteException,
            javax.ejb.DuplicateKeyException,
            javax.ejb.CreateException{
        try {

            MplsVpnServiceKey retKey = null;
```

```
            try {
                retKey =
getSession().createMplsVpnServiceByValue(mplsVpnServiceValue);
            } catch (java.rmi.RemoteException rex) {
                rex.printStackTrace();
                throw new EJBException("RemoteException" + rex.getMessage());
            }

            return retKey;
        } catch (Exception e) {
            String msg = "Unable to createMplsVpnServiceByValue... Exception :  ";
            throw new RemoteException(msg+e);
        }
    }

    @WebMethod(operationName="getMplsVpnServiceByKey")
    public MplsVpnServiceValue  getMplsVpnServiceByKey(MplsVpnServiceKey
mplsVpnServiceKey, String[] attrNames)
    throws javax.oss.OssIllegalArgumentException,
            java.rmi.RemoteException,
            javax.ejb.ObjectNotFoundException{
        try {

            MplsVpnServiceValue retValue = null;
            try {
                retValue = getSession().getMplsVpnServiceByKey(mplsVpnServiceKey,
attrNames);
            } catch (java.rmi.RemoteException rex) {
                rex.printStackTrace();
                throw new EJBException("RemoteException" + rex.getMessage());
            }
            //set all attribute populated for the XML encoding
            ((AttributeAccessImpl)retValue).setFullyPopulated();

            return retValue;
        } catch (Exception e) {
            String msg = "Unable to getMplsVpnServiceByKey... Exception :  ";
            throw new RemoteException(msg+e);
        }
    }

    @WebMethod(operationName="setMplsVpnServiceByValue")
    public void  setMplsVpnServiceByValue(MplsVpnServiceValue mplsVpnServiceValue,
boolean resyncRequired)
    throws javax.oss.OssIllegalArgumentException, java.rmi.RemoteException,
```

```
            javax.ejb.ObjectNotFoundException, javax.oss.OssSetException,
            javax.oss.OssResyncRequiredException{
        try {


            try {
                getSession().setMplsVpnServiceByValue(mplsVpnServiceValue,
resyncRequired);
            } catch (java.rmi.RemoteException rex) {
                rex.printStackTrace();
                throw new EJBException("RemoteException" + rex.getMessage());
            }
        } catch (Exception e) {
            String msg = "Unable to setMplsVpnServiceByValue... Exception :  ";
            throw new RemoteException(msg+e);
        }
    }


    private VpnServiceRemote getSession() throws javax.ejb.CreateException,
javax.naming.NamingException {


        if (initCtx == null) initCtx = new InitialContext();
        vpnHome =
(VpnServiceRemoteHome)initCtx.lookup("System/System1/ApplicationType/Common/Applicat
ion/1-4;1-4;ReferenceImplementation/Comp/VpnServiceBean");
        if (vpnHome == null){
            throw new javax.ejb.CreateException("Lookup of VpnServiceRemoteHome
failed");
        }


        try {
            return vpnHome.create();
        } catch (java.rmi.RemoteException rex) {
            rex.printStackTrace();
            throw new EJBException("RemoteException: " + rex.getMessage());
        }
    }
}
```

The needed annotations are:

- @Stateless: to specify this class is a Stateless Session Bean ;
- @WebService: to indicate JAX-WS this file is a web service, so that artifacts have to be created ;
- @WebMethod: for JAX-WS too, indicating that this method can be request by a web service client ;

Netbeans handles automatically the @WebService annotation by creating the ant target 'wsgen-generate' for producing the endpoint of this web service. As this point wsgen execution should be possible. But JAX-B will encounter obstacle to compute interfaces definitions ; error message like "JAX'B can't handle with interfaces..." will be generated. Next part explains how to handle the generation from interfaces definitions.

Note: when accessing an EJB 2.1 from an EJB 3.0, the object returned by the lookup function can not be casted. The use of an javax.rmi.PortableRemoteObject.narrow() method is mandatory, as explained in the EJB 3.0 specification.

From this interface definition no specific annotation appears for return values and argument of methods. The following section details the preliminary development that have been done in the CBE definitions and implementations.

3.3.2.1 Handling with OSSJ Values and Keys

OSS/J APIs define only interfaces for parameter and return types. JAX-WS tool generates artifacts (java classes) able to convert an xml data file to a java object and vice versa. JAX-WS delegates to JAX-B the binding of xml types to java types. JAX-B needs to instanciate classes (can't be done with interfaces) to encode and decode XML from / to Java .

It may exist several solutions to solve this issue. JAX-B experts have already considered it and propose three solutions. Their three solutions using different annotations:

- @XmlRootElement: implies to annotate all the code, each time when interface is used or implemented.
- @XmlJavaTypeAdapter: only applicable if interface definitions can be modified (regarding JCP licensing model for example)

- @XmlElement: constraint to have a one to one relationship between interfaces and implementations (not always the case for the clients) and cross boundaries between modules can be not supported...

More details can be found in the JAX-B guide, currently written and step by step improved:
https://jaxb.dev.java.net/guide/Mapping_interfaces.html

In our case, the second solution using @XmlJavaTypeAdapter is used for its simplicity and minimal code impact. The @XmlJavaTypeAdapter annotation allows JAX-B to link the XML type to a java encoder/decoder. Its usage involves two constraints :

- annotation of the interface,
- creation of an adapter class with two methods for (un)marshalling between the interface and the implementation class. This avoid the modification of the existing CBE implementation.

Here is an example:

```
package com.infomodel;
   /**
    *    Public interface definition for MplsVpnService
    */
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;
import javax.xml.bind.annotation.XmlType;

@XmlType
@XmlJavaTypeAdapter(value=MplsVpnServiceValueAdapter.class)

public interface MplsVpnServiceValue
  extends javax.oss.cbe.service.ServiceValue
{
  public static final String VALUE_TYPE = "com.infomodel.MplsVpnServiceValue";
  public static final String MPLS_VPN_SERVICE_KEY = "mplsVpnServiceKey";
   public final static String VRF_NAME = "vrfName";

  /** Deep copy of this object  */
    public Object clone();
   /**
    * Attribute getter for vrfName
    * @return the value of the vrfName field.
    * @throws java.lang.IllegalStateException - if the attribute is not populated.
    */
        public java.lang.String getVrfName()
```

```
            throws java.lang.IllegalStateException;


   /**
    * Attribute setter for vrfName
    * @param value - the value to use to set the vrfName attribute.
    * @throws java.lang.IllegalArgumentException - Is thrown to report that a bad
argument was provided to the method.
    */
  public void setVrfName( java.lang.String value )
         throws java.lang.IllegalArgumentException;


    /**
     * Gets the key for this value object
     * @return the key for this value object
     * @throws java.lang.IllegalStateException if no key was populated in this
     *    value object.
     */
  public com.infomodel.MplsVpnServiceKey getMplsVpnServiceKey()
      throws java.lang.IllegalStateException;


  /**
   * Sets the key for this value object
   *
   * @param key - the key to set on this value object.
   * @throws java.lang.IllegalArgumentException, if the key is not a valid
   *    key for this value object.
   */
  public void setMplsVpnServiceKey( com.infomodel.MplsVpnServiceKey key )
         throws java.lang.IllegalArgumentException;


    /**
     * Factory method for MplsVpnServiceKey
     *
     * @return a new instance of a blank MplsVpnServiceKey
     */
  public com.infomodel.MplsVpnServiceKey makeMplsVpnServiceKey();
}
```

...and its Adapter:

```
package com.infomodel;


import javax.xml.bind.annotation.adapters.XmlAdapter;
```

```
public class MplsVpnServiceValueAdapter extends
XmlAdapter<MplsVpnServiceValueImpl,MplsVpnServiceValue>{


    public MplsVpnServiceValue unmarshal(MplsVpnServiceValueImpl p) throws Exception
{
        try {
            MplsVpnServiceValue r = p;
            return r;
        }
        catch (Exception e)
        {
            e.printStackTrace();
            String msg = "Unable to unmarshall MplsVpnServiceValue : "+p;
            throw new Exception(msg+" due to "+e);
        }
    }


    public MplsVpnServiceValueImpl marshal(MplsVpnServiceValue p) throws Exception {
        try {
            MplsVpnServiceValueImpl r = (MplsVpnServiceValueImpl)p;
            return r;
        }
        catch (Exception e)
        {
            e.printStackTrace();
            String msg = "Unable to marshall MplsVpnServiceValueImpl : "+p;
            throw new Exception(msg+" due to "+e);
        }
    }


}
```

Note: One concerns arrays of interfaces. Despite of the previous adaptation, web service artifacts can not be generated. Same error as for a non annotated interface appears: '...JAX-B can't hanlde with interface...'. This bug has been logged on issue tracker on java.net and should be fixed soon.

Workaround : Replace first occurance (of the dependency graph) of each array of interfaces with an array of the corresponding implementation class, and each time this method is called, cast the type in an array of implementation classes.

Note: The adpater classes have been put in the same package as the corresponding implementations. Coding manually the adapters is a tedious task, however they can be easliy generated.

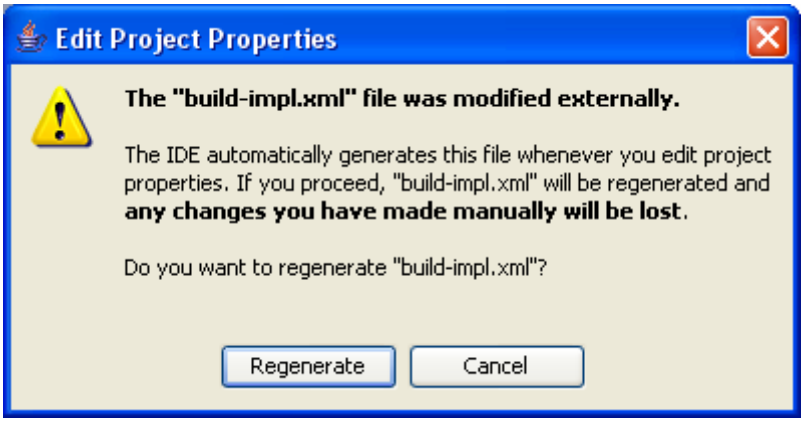
Wsgen target can now be executed again to verify artifacts are well generated; all the previous errors shall disappear. This step is not mandatory, it is just for verification purpose.

At this point of the creation the service is ready to be deployed. Unfortunately, the following error due to a bug need to be workarounded:

```
deployment started : 0%
Deploying application in domain failed; Fatal Error from EJB Compiler --
E:\...j2ee-apps\TroubleTicket2\TTWS2_jar\META-INF\wsdl\JVTTroubleTicketServantService.wsdl
(The system cannot find the file specified)
```

The automatically (by netbeans) generated ant rules in the project management files needs to be updated in order to run wsgen and package the generated WSDL into the archive.

Netbeans will detect that its internal project file have been modified, and will pops up:



Always select “Cancel” to preserve the following changes in build-impl.xml of the project (follow the “VP” sections):

Clean section:

```
    ===============

    CLEANUP SECTION

    ===============

    -->

    <target name="deps-clean" depends="init" if="no.dist.ear.dir" unless="no.deps">

        <ant target="clean" inheritall="false"
antfile="${project.common_ex_EJB}/build.xml"/>
```

```
    </target>
    <target name="-do-clean" depends="init">
        <delete dir="${build.dir}"/>
        <delete dir="${dist.dir}"/>
        <!-- VP -->
        <delete dir="${src.dir}/../conf/wsdl"/>
        <!-- end VP -->
    </target>
```

Dist section:

```
    ====================
    DIST BUILDING SECTION
    ====================
    -->
    <target name="-pre-dist">
        <!-- Empty placeholder for easier customization. -->
        <!-- You can override this target in the ../build.xml file. -->
        <!-- VP -->
        <ejbjarproject2:javac srcdir="${build.generated.dir}/wsgen/service"
destdir="${build.classes.dir}"/>
        <ejbjarproject2:javac srcdir="${build.generated.dir}/wsgen/service"
destdir="${build.ear.classes.dir}"/>
        <!-- end VP -->
…
    <!-- VP add wsgen-generate after compile target in both following  depends list
-->
    <target name="dist" depends="init,compile,wsgen-generate,-pre-dist,-do-dist,-
post-dist" description="Build distribution (JAR)."/>
    <target name="dist-ear" depends="init,compile,wsgen-generate,-pre-dist,-do-ear-
dist,-post-dist" description="Build distribution (JAR) to be packaged into an
EAR."/>
    <!-- end VP -->
```

Compilation section:

```
     ====================
    COMPILATION SECTION
    ====================
    -->
    <target name="-deps-module-jar" depends="init" if="no.dist.ear.dir"
unless="no.deps">
        <ant target="dist" inheritall="false"
antfile="${project.common_ex_EJB}/build.xml"/>
    </target>
    <target name="-deps-ear-jar" depends="init" if="dist.ear.dir" unless="no.deps">
        <ant target="dist-ear" inheritall="false"
antfile="${project.common_ex_EJB}/build.xml">
```

```
            <property name="dist.ear.dir" location="${build.dir}"/>
        </ant>
    </target>
    <target name="deps-jar" depends="init, -deps-module-jar, -deps-ear-jar"/>
    <target name="wsgen-init" depends="init">
        <mkdir dir="${build.generated.dir}/wsgen/service"/>
        <mkdir dir="${classes.dir}"/>
        <taskdef name="wsgen" classname="com.sun.tools.ws.ant.WsGen">
            <classpath path="${j2ee.platform.wsgen.classpath}"/>
        </taskdef>
        <!-- VP -->
        <mkdir dir="${src.dir}/../conf/wsdl"/>
        <!-- end VP -->
    </target>
    <target name="wsgen-VpnService" depends="wsgen-init, compile">
        <wsgen sourcedestdir="${build.generated.dir}/wsgen/service"
resourcedestdir="${build.generated.dir}/wsgen/service" keep="true" genwsdl="true"
sei="ossj.common.ex.VpnService">
            <classpath
path="${classes.dir}:${j2ee.platform.wsgen.classpath}:${javac.classpath}"/>
        </wsgen>
        <!-- VP copy the files where needed for jar packaging -->
        <copy todir="${src.dir}/../conf/wsdl">
            <fileset dir="${build.generated.dir}/wsgen/service" includes="*.wsdl
*.xsd"/>
        </copy>
        <!-- end VP -->
```

Now, the project shall be able to deploy successfully from the application project using the “Run Project” item (right-click on the project.

The deployment could be easily verified by accessing the wsdl file of the web browser using the following URL:

http://localhost:8080/VpnServiceService/VpnService?WSDL

The web service interface could also being test using an integrated client application using the same EJB module in netbeans.

A specific client example is provided with this project to show how to handle complex types. This client application needs 2 inputs:

- the URL of the wsdl file of the service to test (see below). Service description is used by wsimport ot generate the client WS artifacts;

- class definitions of the specific types used.

The client retrieves service port information from wsdl file, then invokes the exposed services.

The client code is cided in the “Test packages” section of the EJB module project. This simplify the configuration effort.

Find below the code of this client application:

```
package client;
import javax.xml.soap.SOAPFault;
import javax.xml.ws.WebServiceRef;
import javax.xml.ws.soap.SOAPFaultException;

import ossj.common.ex.MplsVpnServiceKeyImpl;
import ossj.common.ex.MplsVpnServiceValueImpl;
import ossj.common.ex.VpnService;
import ossj.common.ex.VpnServiceService;

public class VpnServiceClient {
    public VpnServiceClient() {
    }

@WebServiceRef(wsdlLocation="http://localhost:8080/VpnServiceService/VpnService?WSDL
")
    static VpnServiceService service;
    public static void main(String[] args) {
        try {
            VpnServiceClient  client = new VpnServiceClient();
            client.doTest(args);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    public void doTest(String[] args) {
        try {
            System.out.println(" Retrieving port from the service " + service);
            VpnService port = service.getVpnServicePort();

            System.out.println(" Invoking 'createMplsVpnService' operation on the
Servant port");
            MplsVpnServiceValueImpl mplsVpn = new MplsVpnServiceValueImpl();
            //set the mandatory attributes
            mplsVpn.setVrfName("myVrfName");
            mplsVpn.setSubGraphId(0);
            mplsVpn.setState("Active");
            mplsVpn.setSubscriberId("mySubscriberId");
```

```
            mplsVpn.setMandatory(false);
            mplsVpn.setStartMode(0);


//invoke create method on the WS
            MplsVpnServiceKeyImpl mplsVpnKey =
(MplsVpnServiceKeyImpl)port.createMplsVpnServiceByValue(mplsVpn);


            System.out.println(" Result, MplsVpnServiceKey returned for the new
created:");
            System.out.println("    primaryKey: "+mplsVpnKey.getPrimaryKey());
            System.out.println("    type      : "+mplsVpnKey.getType());
            System.out.println("    VrfName   : "+mplsVpn.getVrfName());


// get the new created vpnservice
            System.out.println(" Invoking 'getMplsVpnService' with
"+mplsVpnKey.getPrimaryKey());
            MplsVpnServiceValueImpl mplsValue =
(MplsVpnServiceValueImpl)port.getMplsVpnServiceByKey(mplsVpnKey,null);
            System.out.println(" Result, MplsVpnServiceKey returned for the new
created:");
            System.out.println("    primaryKey    :
"+mplsValue.getMplsVpnServiceKey().getPrimaryKey());
            System.out.println("    SubsbcriverId : "+mplsValue.getSubscriberId());
            System.out.println("    VrfName       : "+mplsValue.getVrfName());


// change some values
            System.out.println(" Invoking 'setMplsVpnServiceByValue'");
            mplsValue.setSubscriberId("mySubscriberId");
            mplsValue.setVrfName("updated VrfName:
["+mplsVpnKey.getPrimaryKey()+"]");
            port.setMplsVpnServiceByValue(mplsValue,false);


// get the new created vpnservice
            System.out.println(" Invoking 'getMplsVpnService' with
"+mplsVpnKey.getPrimaryKey());
            mplsValue =
(MplsVpnServiceValueImpl)port.getMplsVpnServiceByKey(mplsVpnKey,null);
            System.out.println(" Result, MplsVpnServiceKey returned for the new
created:");
            System.out.println("    primaryKey    :
"+mplsValue.getMplsVpnServiceKey().getPrimaryKey());
            System.out.println("    SubsbcriverId : "+mplsValue.getSubscriberId());
            System.out.println("    VrfName       : "+mplsValue.getVrfName());


        } catch(SOAPFaultException Se) {
            SOAPFault fault = Se.getFault();
            System.out.print("SOAPFaultException thrown while running the client :
"+fault.toString());
```

```
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

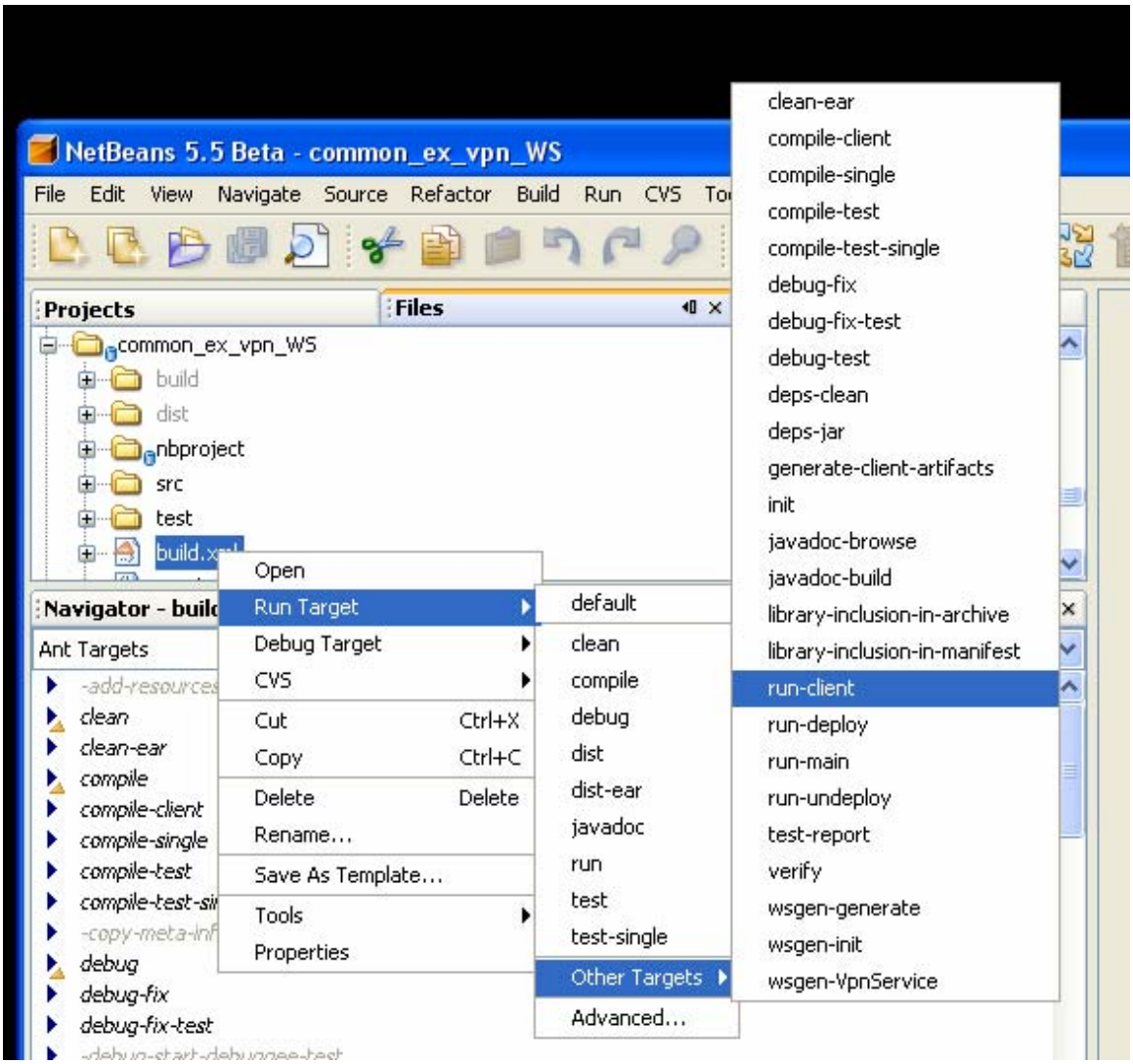
The project build-impl.xml have been updated to support correctly the client compilation and execution:

```
    <!-- VP -->

    <!--
    ========================
    TEST WITH CLIENT SECTION
    ========================
    -->
    <property name="WSIMPORT" value="${com.sun.aas.installRoot}/bin/wsimport.bat"/>
    <property name="APPCLIENT"
value="${com.sun.aas.installRoot}/bin/appclient.bat"/>
    <property name="http.port" value="8080"/>
    <property name="http.host" value="localhost"/>

    <target name="compile-client" depends="init, generate-client-artifacts">
      <javac srcdir="${test.src.dir}" destdir="${build.classes.dir}/client"

classpath="${build.classes.dir}:${javac.classpath}:${j2ee.platform.classpath}"
        includes="client/**"/>
    </target>

    <target name="generate-client-artifacts" depends="init">
      <mkdir dir="${build.classes.dir}/client"/>
      <echo message="${WSIMPORT} -keep -d ${build.classes.dir}/client
    http://${http.host}:${http.port}/VpnServiceService/VpnService?WSDL"/>
      <exec executable="${WSIMPORT}" failonerror="true" >
        <arg line="-keep -d ${build.classes.dir}/client
    http://${http.host}:${http.port}/VpnServiceService/VpnService?WSDL"/>
      </exec>
    </target>

    <target name="run-client" depends="init">
     <echo message="Executing appclient with client class as
client.VpnServiceClient"/>
     <exec executable="${APPCLIENT}" dir="${build.classes.dir}/client">
        <arg value="client.VpnServiceClient"/>
      </exec>
```

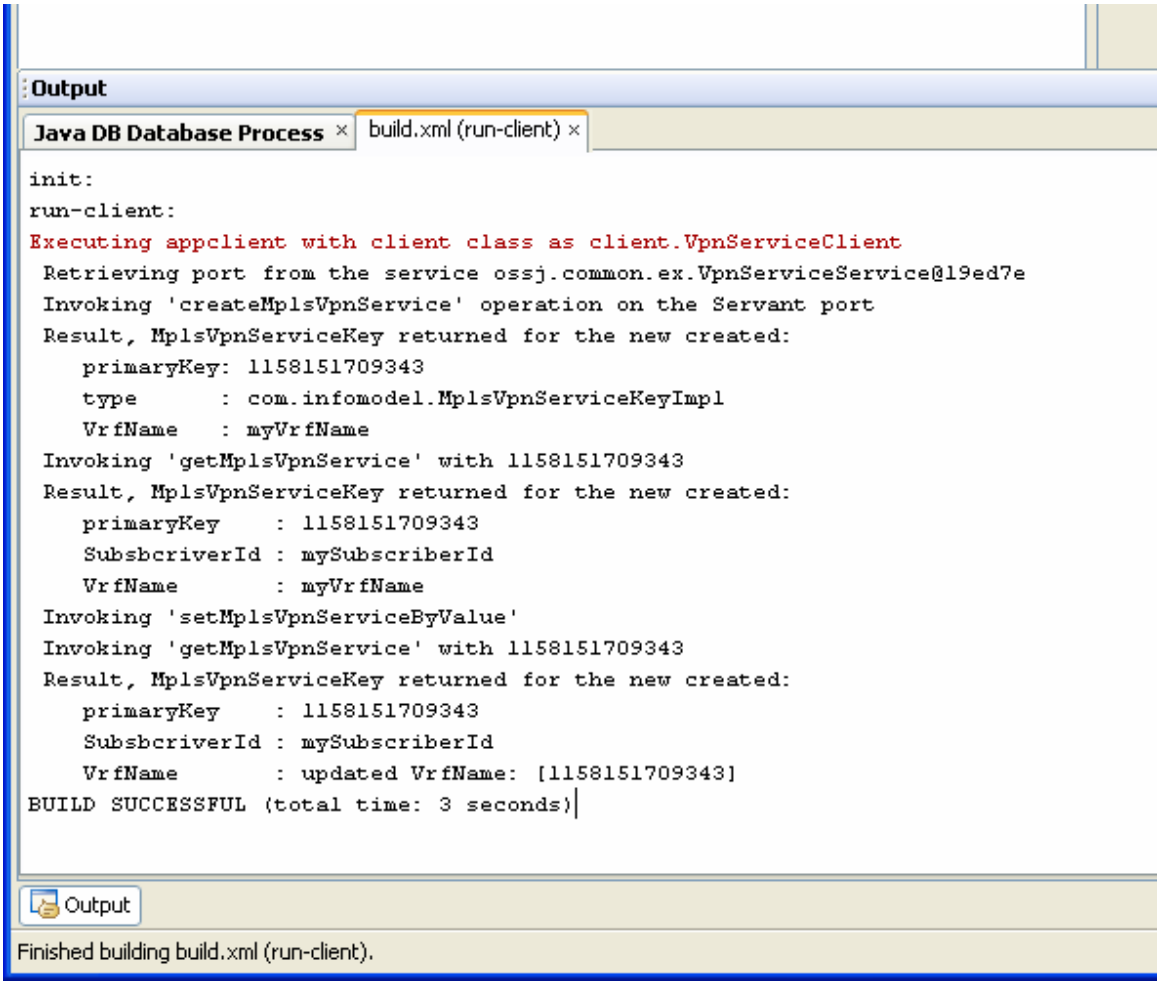
```
    </target>
    <!-- end VP -->
```

In order to compile and execute the client, call the “compile-client” and run-client target from the contextual menu of the project:

In netbeans, select the “Files” section, open the Project directory right-click on the build.xml and select the “compile-client” then the “run-client targets.



At the bottom section of netbeans window, you will see the execution output as follow:



3.4 Verify the Application

According to the OSS/J recommendation, the application have to be ferifed using the Java EE verifier. This tool is integrated into the NetBeans and a downloadable plugin.

Using the Update Center download and install the Sun Java System Application Verification Kit (AVK).

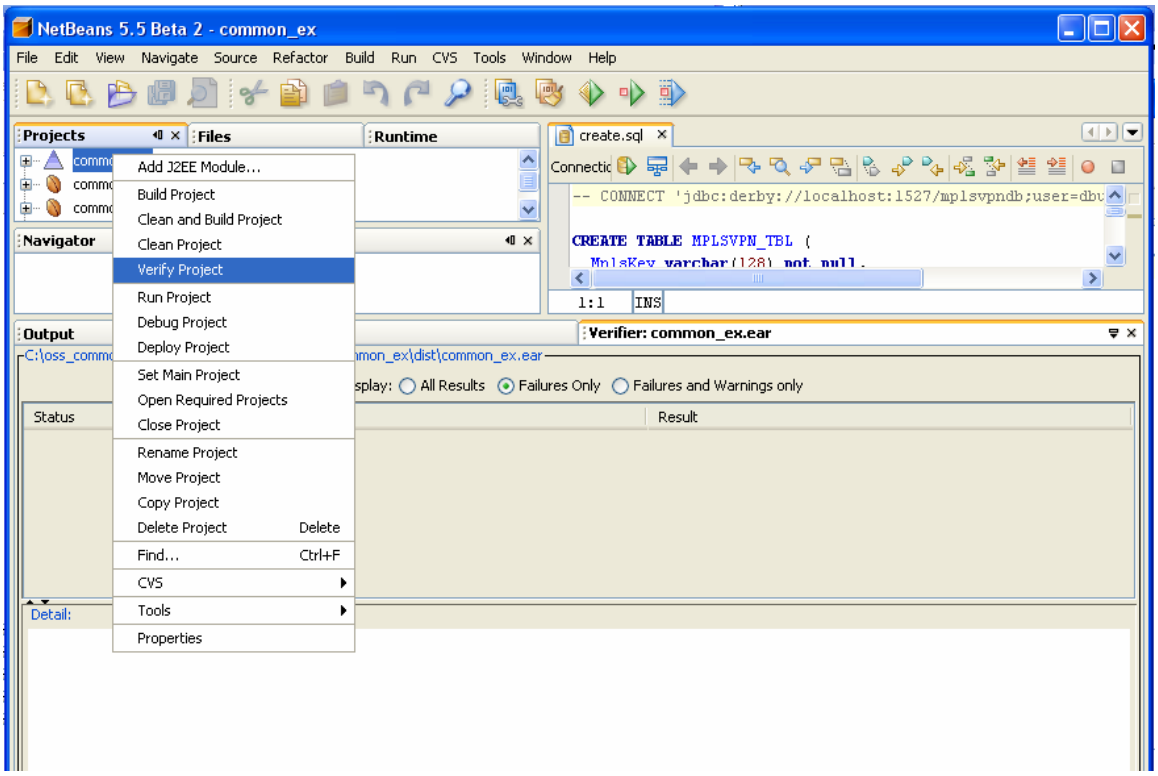
And download the latest AVK for JavaEE 5 from

http://java.sun.com/javaee/downloads/index.jsp

(Note: point the installation wizard to the Application Server already installed)

3.4.1 Static verification

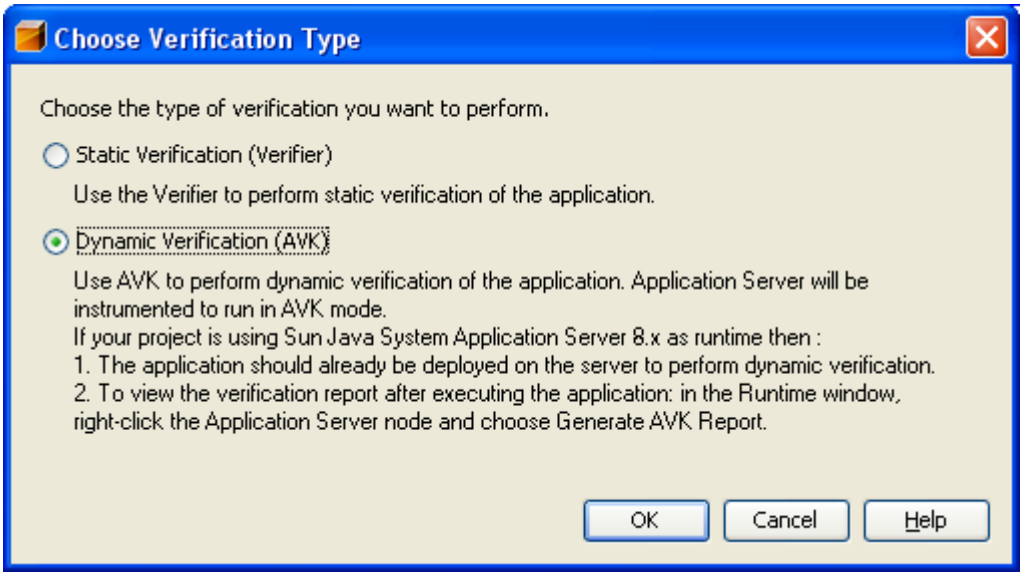
Select the common_ex project and select the “verify” item:



Select first the Static verification:

The common_ex application contains no error.

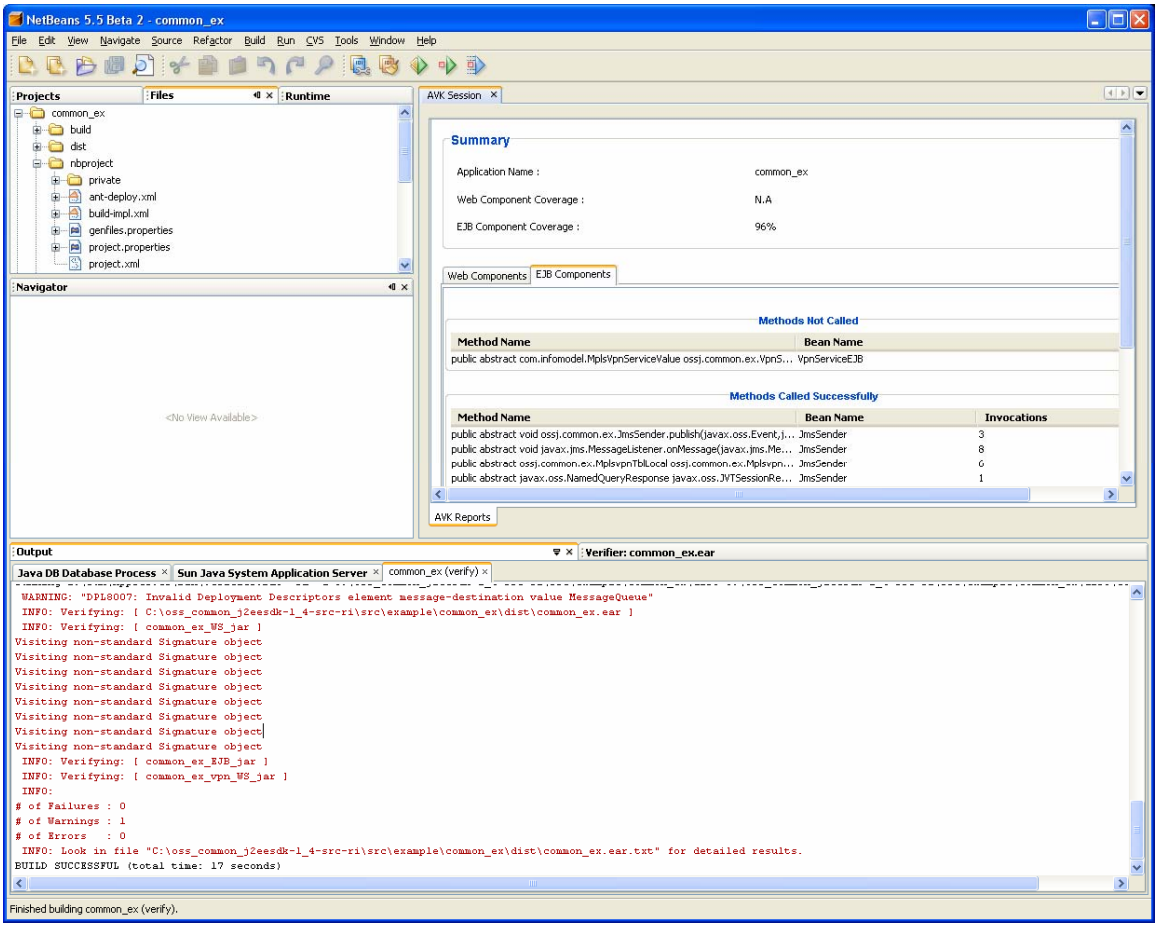
Repeat this step and then select the Dynamic Verification:



And follow instruction.

Execute the TCKs, once terminated, to view the verification report after executing the application: in the Runtime window, right-click the Application Server node and choose Generate AVK Report.

The AVK reports for Common RI 96% coverage => successfully pass the test



Appendix A: Glossary and References

References