Oracle® Tuxedo

Accessing Mainframe from Java

12*c* Release 2 (12.1.3)

April 2014

ORACLE

Oracle Tuxedo Accessing Mainframe from Java, 12*c* Release 2 (12.1.3)

Contents

Generating a Java Application with the eGen Application Generator

Tuxedo Mainframe Transaction Publisher

Generating a Java Application with the eGen Application Generator

This document includes the following topics:

Overview

Oracle Tuxedo supports seamless integration of CICS Transaction Gateway (CTG) application running on J2EE application servers and JCA based.

With this feature, Oracle Tuxedo provides a tool to

- parse COBOL copybooks used to describe CICS transactions/programs interfaces
- generate Java bean style classes to populate data

Therefore, users can pass those classes to a CCI (or ECI-wrapped) interface to perform ART-hosted CICS invocations.

Writing an eGen Script

After you have obtained a COBOL Copybook for the mainframe applications, you are ready to write an eGen script. This eGen script and the COBOL copybook that describes your data structure will be processed by the eGen utility to generate a DataView and application code which will serve as the basis for your custom Java application.

An eGen script has two sections. These are:

- DataView. The DataView section of the script generates Java DataView code from a COBOL copybook. The class file compiled from the generated code extends the Java DataView class. Generating DataViews is discussed in detail in the remainder of this section.

  Note: If the purpose of your eGen script is to generate a DataView for use with the WebLogic JAM to JMS EJB, or to launch a WebLogic Integration event, you only need to create the DataView section of the script.

- Java application. The Java application section of the script generates the Java application code. This is discussed in detail in Basic Programming Techniques.

Writing the DataView Section of an eGen Script

The eGen utility parses a COBOL copybook and generates Java DataView code that encapsulates the data record declared in the copybook. It does this by parsing an eGen script file containing a DataView definition similar to the example shown in Listing 1 (keywords are in bold). The section containing the DataView definition is the first section of the eGen script. Application code is generated by the second section.

Listing 1 Sample DataView Section of an eGen Script

```
 generate view examples.CICS.outbound.gateway.EmployeeRecord from
emprec.cpy
```

Analyzing the parts of this line of code, we see that **`generate view`** tells the eGen utility to generate a Java DataView code file. `examples.CICS.outbound.gateway.EmployeeRecord` tells the eGen utility to call the DataView file `EmployeeRecord.java`. The package is called `examples.CICS.outbound.gateway`. The `EmployeeRecord` class defined in

`EmployeeRecord.java` is a subclass of the DataView class. The phrase from `emprec.cpy` tells the eGen utility to form the `EmployeeRecord` DataView file from the COBOL copybook `emprec.cpy`.

Additional `generate view` statements may be added to an eGen script in order to produce all the DataViews required by your application. Also, additional options may be specified in the eGen script to change details of the DataView generation. For example, the following script will generate a DataView class that uses codepage `cp500` for conversions to and from mainframe format. If the codepage clause is not specified, the default codepage of `cp037` is used.

Listing 2 Sample DataView Section with Codepage Specified

```
 generate view examples.CICS.outbound.gateway.EmployeeRecord from
emprec.cpy codepage cp500
```

The following script will generate additional output intended to support use of the DataView class with XML data:

Listing 3 Sample DataView Section Supporting XML

```
generate view sample.EmployeeRecord from emprec.cpy support xml
```

Additional files generated for XML support are listed in Table 1.

Table 1 Additional Files for DataView XML Support

File Name	File Purpose
classname.dtd	XML DTD for XML messages accepted and produced by this DataView.
classname.xsd	XML schema for XML messages accepted and produced by this DataView.

Field Name Mapping Rules

When you process a COBOL copybook containing field names, they are mapped to Java names by the eGen utility. All alphabetic characters are mapped to lower case, except in the following two cases.

All dashes are removed and the character following the dash is mapped to upper case.

When a prefix is added to the name (as when creating a field accessor function name), the first character of the base name is mapped to upper case.

Table 2 lists some mapping examples.

Table 2 Example Field Name Mapping from COBOL to Java and Accessor

COBOL Field Name	Java Base Name	Sample Accessor Name
EMP-REC	empRec	setEmpRec
500-REC-CNT	500RecCnt	set500RecCnt

Field Type Mappings

When you process a COBOL copybook, the data types of fields are mapped to Java data types. The mapping is performed by the eGen utility according to the following rules:

1. Groups map to DataView subclasses.
2. All alphanumeric fields are mapped to type String.
3. All edited numeric fields are mapped to type String.
4. All SIGN SEPARATE, BLANK WHEN ZERO or JUSTIFIED RIGHT fields are mapped to type String.
5. SIGN IS LEADING is not supported.
6. The types COMP-1, COMP-2, COMP-5, COMP-X, and PROCEDURE-POINTER fields are not supported (an error message is generated).
7. All INDEX fields are mapped to Java type int.
8. POINTER maps to Java type int.

9. All numeric fields with any digits to the right of the decimal point are mapped to type BigDecimal.
10. All COMP-3 (packed) fields are mapped to type BigDecimal.
11. All other numeric fields are mapped as shown in Table 3.

Table 3 Numeric Field Mapping

Number of Digits	Java Type
<= 4	short
> 4 and <= 9	int
> 9 and <= 18	long
> 18	BigDecimal

Accessors

This topic includes the following parts.

- Group Field Accessors
- Elementary Field Accessors
- Array Field Accessors
- Fields with REDEFINES Clauses

Group Field Accessors

Each nested group in a COBOL copybook is mapped to a corresponding `DataView` subclass. The generated subclasses are nested exactly as the COBOL groups in the copybook. In addition, the eGen utility generates a private instance variable of this class type and a `get` accessor.

For example, the following copybook:

```
10 MY-RECORD.
       20  MY-GRP.
              30  ALNUM-FIELD            PIC X(20).
```

Produces code similar to the following:

```
 public MyGrp2V getMyGrp();
public static class MyGrp2V extends DataView
{
              // Class definition
}
```

Elementary Field Accessors

Each elementary field is mapped to a private instance variable within the generated DataView subclass. Access to this variable is accomplished by two accessors that are generated (`set` and `get`).

These accessors have the following forms:

```
 public void setFieldName(FieldType value);
 public FieldType getFieldName();
```

Where:

`FieldType`

is described in the Field Type Mappings section.

`FieldName`

is described in the Field Name Mapping Rules section.

For example, the following copybook:

```
 10 MY-RECORD.
       20  NUMERIC-FIELD                PIC S9(5).
       20  ALNUM-FIELD                  PIC X(20).
```

Produces the accessors:

```
public void setNumericField(int value);
public int getNumericField();
public void setAlnumField(String value);
public String getAlnumField();
```

Array Field Accessors

Array fields are handled according to the field accessor rules described in Group Field Accessors and Elementary Field Accessors, with the addition that each accessor takes an additional `int` argument that specifies which array entry is to be accessed, for example:

```
public void          setFieldName(int index, FieldType value);

public FieldType     getFieldName(int index);
```

Array fields specified with the `DEPENDING ON` clause are handled the same as fixed-size arrays with the following special rules:

- The accessors may be used to `get` or `set` any instance up to the maximum array index.
- The controlling (`DEPENDING ON`) variable is evaluated when the DataView is converted to or from an external format, such as a mainframe format. The eGen utility converts only the array elements with subscripts less than the controlling value.

Fields with REDEFINES Clauses

Fields that participate in a `REDEFINES` set are handled as a unit. A private `byte[]` variable is declared to hold the underlying mainframe data, as well as a private `DataView` variable. Each of the redefined fields has an accessor or accessors. These accessors take more CPU overhead than the normal accessors because they perform conversions to and from the underlying `byte[]` data.

For example the copybook:

```
 10 MY-RECORD.
       20 INPUT-DATA.
             30 INPUT-A                              PIC X(4).
             30 INPUT-B                              PIC X(4).
       20 OUTPUT-DATA REDEFINES INPUT-DATA           PIC X(8).
```

Produces Java code similar to the following:

```
private byte[] m_redef23;

private DataView m_redef23DV;

public InputDataV   getInputData();

public String getOutputData();

public void   setOutputData(String value);
```

```
public static class InputDataV extends DataView
{
// Class definition.
}
```

COBOL Data Types

This section summarizes the COBOL data types supported by WebLogic JAM software. Table 4 lists the COBOL data item definitions recognized by the eGen utility. Table 5 lists the syntactical features and data types recognized by the eGen utility. If a COBOL feature is unsupported and it is not listed as ignored in the table, an error message is generated.

Table 4 Major COBOL Features

COBOL Feature	Support
IDENTIFICATION DIVISION	Unsupported
ENVIRONMENT DIVISION	Unsupported
DATA DIVISION	Partially Supported
WORKING-STORAGE SECTION	Partially Supported
Data record definition	Supported
PROCEDURE DIVISION	Unsupported
COPY	Unsupported
COPY REPLACING	Unsupported
EJECT, SKIP1, SKIP2, SKIP3	Supported

Table 5 COBOL Data Types

COBOL Type	Java Type
COMP, COMP-4, BINARY (integer)	Short/Int/Long
COMP, COMP-4, BINARY (fixed)	BigDecimal

Table 5 COBOL Data Types

COBOL Type	Java Type
COMP-3, PACKED-DECIMAL	BigDecimal
COMP-5	Unsupported
COMP-X	Unsupported
DISPLAY numeric (zoned)	BigDecimal
BLANK WHEN ZERO (zoned)	String
SIGN IS LEADING (zoned)	Unsupported
SIGN IS LEADING SEPARATE (zoned)	String
SIGN IS TRAILING (zoned)	String
SIGN IS TRAILING SEPARATE (zoned)	String
edited numeric	String
COMP-1, COMP-2 (float)	Unsupported
edited float numeric	String
DISPLAY (alphanumeric)	String
edited alphanumeric	String
INDEX	Int
POINTER	Int
PROCEDURE-POINTER	Unsupported
JUSTIFIED RIGHT	Unsupported (ignored)
SYNCHRONIZED	Unsupported (ignored)
REDEFINES	Supported
66 RENAMES	Unsupported
66 RENAMES THRU	Unsupported
77 level	Supported

Table 5 COBOL Data Types

COBOL Type	Java Type
88 level (condition)	Unsupported (ignored)
group record	Inner Class
OCCURS (fixed array)	Array
OCCURS DEPENDING (variable-length array)	Array
OCCURS INDEXED BY	Unsupported (ignored)
OCCURS KEY IS	Unsupported (ignored)

Program Development

Program development will be accomplished according to program snippet listed in Listing 4 and according to class naming rules outlined here, although this can be adjusted depending on customer requirements.

Listing 4 Program Snippet

```
try
     {
        InitialContext context = new InitialContext();

       ECIConnectionSpec connSpec = new ECIConnectionSpec();
       connSpec.setUserName("TESOP01");
       connSpec.setPassword("");
       Connection connection = connectionFactory.getConnection(connSpec);

        Interaction interaction = connection.createInteraction();

        // Create inputBean
```

```
                K294Bean inRec = new K294Bean();
                inRec.setI__Entete__TranId("K294");
                inRec.setI__Entete__Vers("0101");
                inRec.setI__Entete__Statut("99");
                inRec.setI__Entete__Nb__Enreg((short)40);
                inRec.setI__Entete__User("TESOP01");
                inRec.setI__Entete__Date("2012-01-16");

        // Data
                inRec.setI__restea__nupy(1);
                inRec.setI__restea__cdea(2);
                inRec.setI__restea__cdea1(1);

                K294Bean outRec = new K294Bean();

      // Create InteractionSpec
        InteractionSpec interactionSpec = new ECIInteractionSpec();
        ((ECIInteractionSpec)interactionSpec).setFunctionName("COMPT294");
        ((ECIInteractionSpec)interactionSpec).setTranName("K294");
        ((ECIInteractionSpec)interactionSpec).setCommareaLength(7132);

((ECIInteractionSpec)interactionSpec).setInteractionVerb(ECIInteractionSpe
c.SYNC_SEND_RECEIVE);

        // execute transaction
        interaction.execute((ECIInteractionSpec)interactionSpec, inRec,
outRec);
```

```
        // Close all
                interaction.close();
                connection.close();


       // List Data
           K294bean_output__message_t__o__data__data  data[] =
outRec.getT__o__data__data();


      // Load List
      for (int i=0; i<data.length;i++)
      {
               if (data[i].getT__o__data__data__o__restea__cdea()!=0)
                 {
                   out.println(data[i]);
                 }
      }
  }
catch (Exception e)
      {
        System.out.println("Error : " + e.getMessage());
        e.printStackTrace();
      }
```

Important Areas

The following listings show the important areas for program development. Field name mappings may vary.

- Listing 5, “Setup Connection,” on page -13

Listing 5 Setup Connection

```
ECIConnectionSpec connSpec = new ECIConnectionSpec();
       connSpec.setUserName("TESOP01");
       connSpec.setPassword("");
       Connection connection = connectionFactory.getConnection(connSpec);

        Interaction interaction = connection.createInteraction();
  // Create InteractionSpec
        InteractionSpec interactionSpec = new ECIInteractionSpec();
        ((ECIInteractionSpec)interactionSpec).setFunctionName("COMPT294");
        ((ECIInteractionSpec)interactionSpec).setTranName("K294");
        ((ECIInteractionSpec)interactionSpec).setCommareaLength(7132);
((ECIInteractionSpec)interactionSpec).setInteractionVerb(ECIInteractionSpe
c.SYNC_SEND_RECEIVE);
```

Listing 6 Input Bean Usage

```
// Create inputBean
                K294Bean inRec = new K294Bean();
                inRec.getDfhcommarea().
                    getInputMessage().
                    getIEntete().setIEnteteTranId("K294");
                inRec.getDfhcommarea().
```

```
                    getInputMessage().
                    getIEntete().setIEnteteVers("0101");
                inRec.getDfhcommarea().
                    getInputMessage().
                    getIEntete().setIEnteteStatut("99");
                inRec.getDfhcommarea().
                    getInputMessage().
                    getIEntete().setIEnteteNbEnreg((short)40);
// reserve outputBean
          K294Bean outRec = new K294Bean();
```

Listing 7 Service Invocation

```
// execute transaction
        interaction.execute((ECIInteractionSpec)interactionSpec, inRec,
outRec);
```

Listing 8 Output Bean Usage

```
K294bean_output__message_t__o__data__data  data[] =
outRec.getDfhcommarea().getOutputMessage().getTODataData();
```

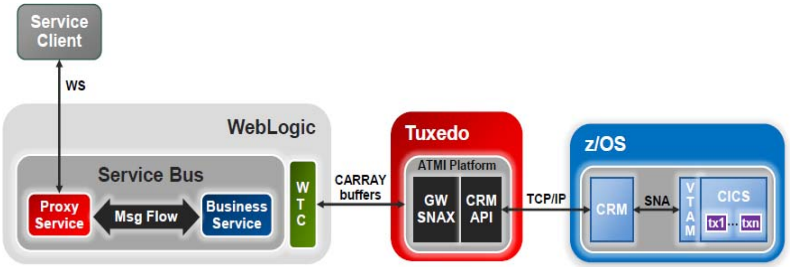
Tuxedo Mainframe Transaction Publisher

This document includes the following topics:

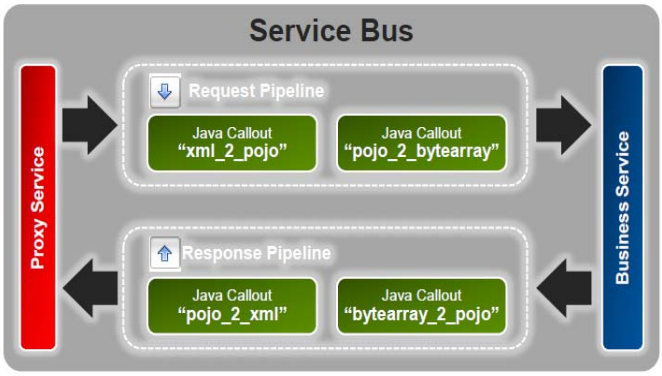
Overview

Tuxedo Mainframe Transaction Publisher simplifies the process of exposing mainframe transaction in Oracle Service Bus (OSB) by providing a graphical user interface.

Let us consider this scenario, where users want to expose their mainframe transaction in OSB. The proxy service uses WSDL and the business service uses WTC.



The tool generates POJO code based on the input COBOL copybook. These generated codes can be used by users to access mainframe transaction.

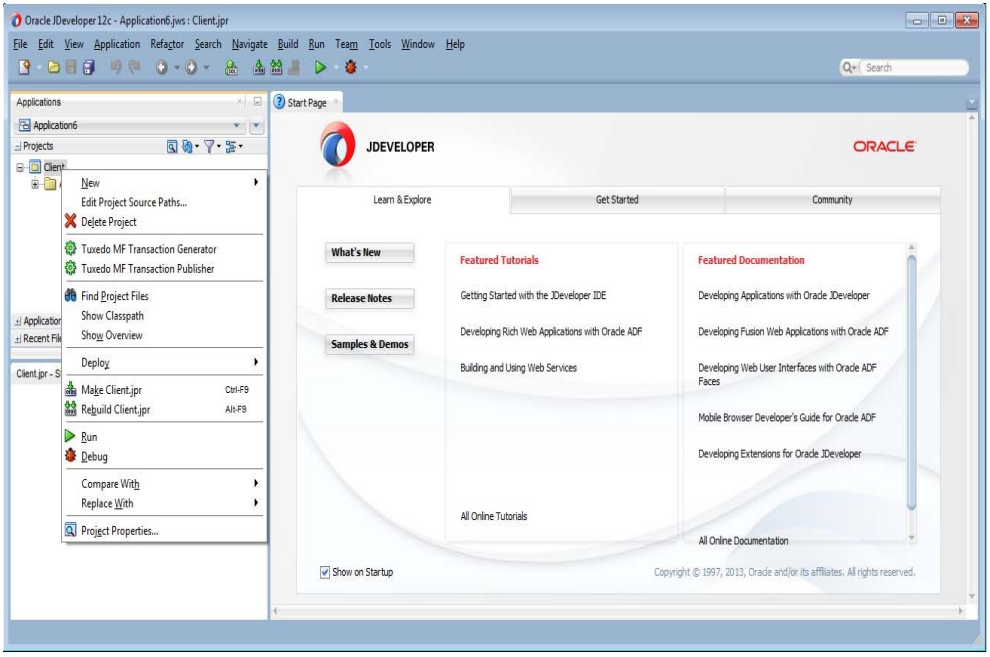


Using Tuxedo Mainframe Transaction Publisher

Tuxedo Mainframe Transaction Publisher includes two parts: Generator and Publisher. They are implemented as JDeveloper extensions and reside in a single JAR file.

- Tuxedo Mainframe Transaction Generator
- Tuxedo Mainframe Transaction Publisher

Tuxedo Mainframe Transaction Publisher is a project based tool. Users select the project and right click to bring up context menu.



Note: Users install this extension using JDeveloper's update center mechanism. For more information, see Installing Tuxedo Mainframe Transaction Publisher.

Tuxedo Mainframe Transaction Generator

Tuxedo Mainframe Transaction Generator is implemented through the JDeveloper hook. Users access this function by clicking the "Tuxedo Mainframe Transaction Generator" menu item.

By selecting this function, a graphical user interface base wizard window will be brought up to guide users to do the following things.

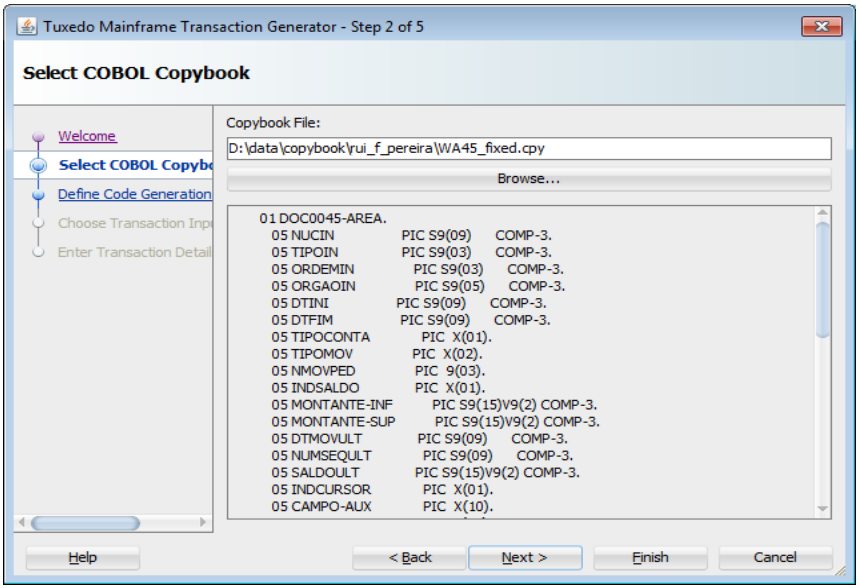
1. Select COBOL Copybook
2. Define Code Generation Details
3. Configure Transaction Input and Output
4. Enter Transaction Details

Eventually, Tuxedo Mainframe Transaction Generator generates seven artifacts that are organized in two parts.

- Generated Java code based on the COBOL copybook
- OSB related configuration data which includes WSDL, configuration for OSB Business Service, and configuration information for OSB Proxy Service

Select COBOL Copybook

The following picture shows the wizard page for selecting COBOL copybook.



Define Code Generation Details

The following screenshot shows the wizard page for defining code generation details.

The following fields are used.

Transaction ID

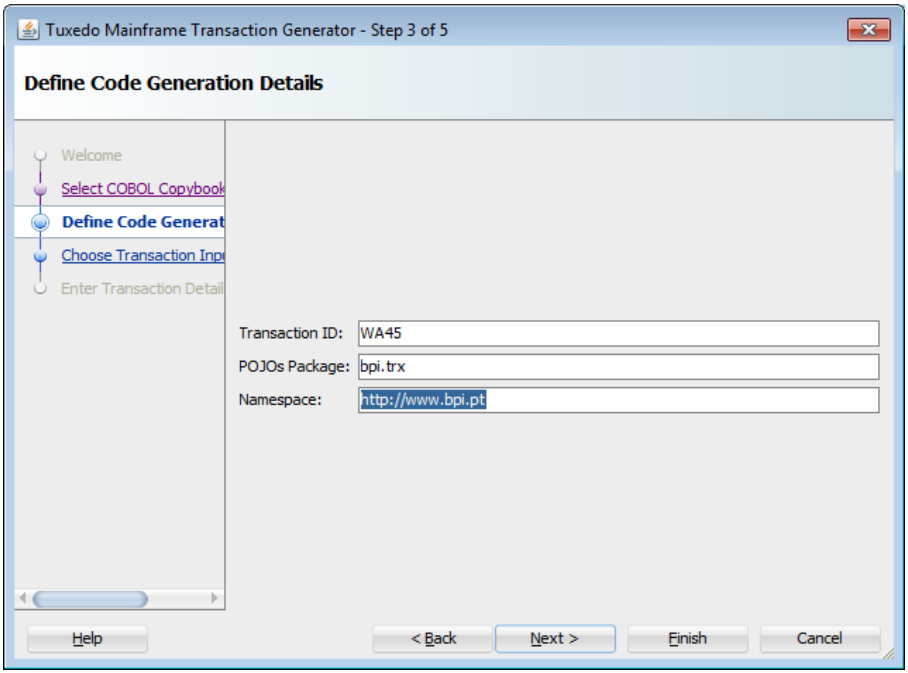
Name of the mainframe transaction. This is used in code and artifacts generation to name the OSB project, artifacts, and data mapping classes.

POJOs Package

This is used as Java package name for the mapping classes.

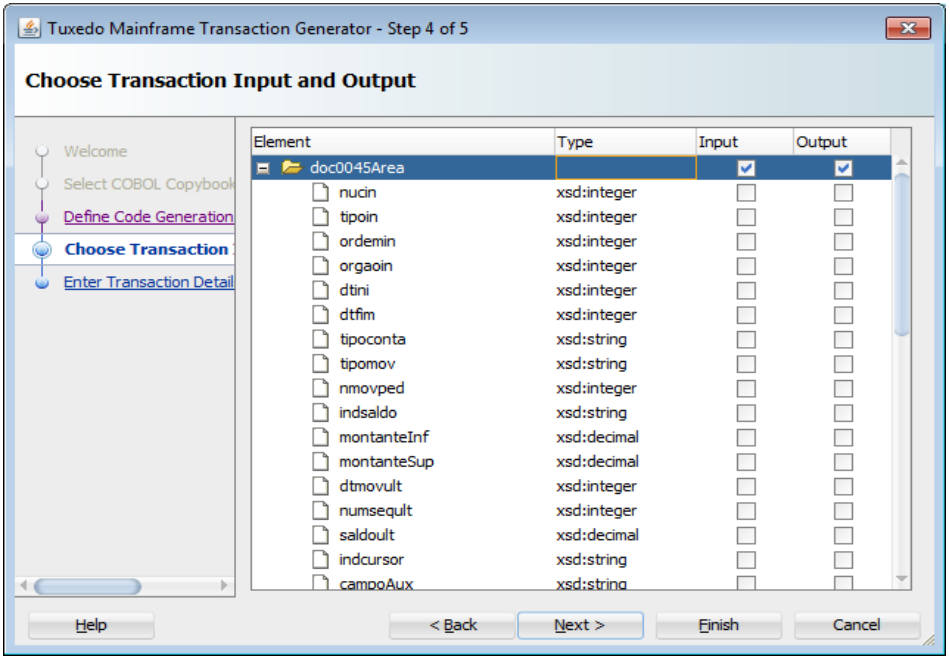
Namespace

This is used as WSDL and schema namespace in the WSDL and XSD OSB artifacts.



Configure Transaction Input and Output

The following screenshot shows the wizard page for configuring the input and output fields from the COBOL copybook.



Enter Transaction Details

The following screenshot shows wizard page for entering information needed by mainframe transaction.

The following fields are used.

Tuxedo transaction resource name
: Name of the generated Tuxedo transport/WTC import that will be generated.

Tuxedo transaction remote name
: Name of the Tuxedo service on the remote Tuxedo domain as exported from there.

Tuxedo remote domain
: ID of the remote Tuxedo/TMA domain.

Tuxedo network address
: Network address for the Tuxedo/TMA remote domain.

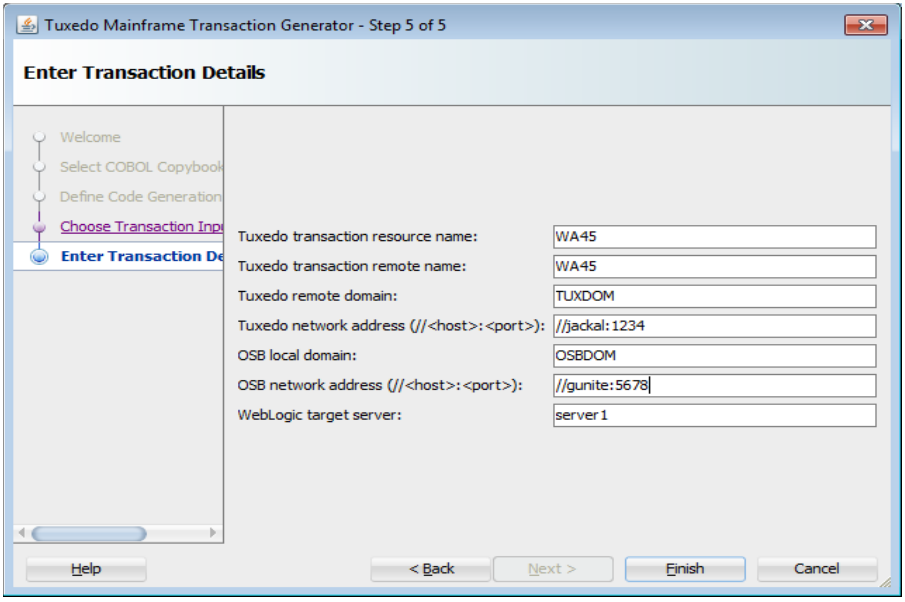
OSB local domain
: ID of the OSB domain.

OSB network address

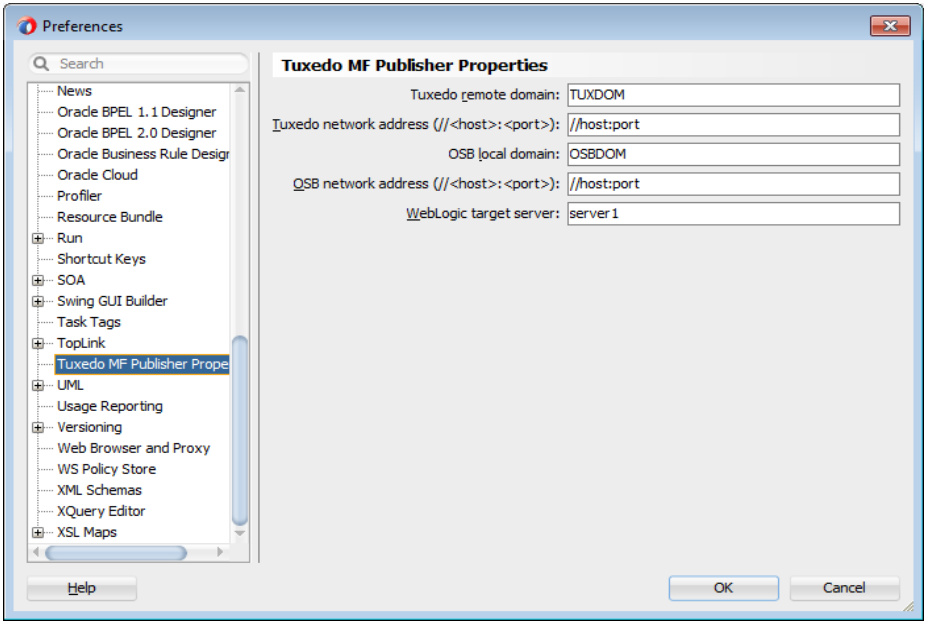
Network address of the OSB domain.

WebLogic target server

Name of the WLS server.



Users are allowed to set the defaults value for the mainframe transaction details according to user needs through the JDeveloper's "Preference" menu item from the "Tools" drop down menu.



Tuxedo Mainframe Transaction Publisher

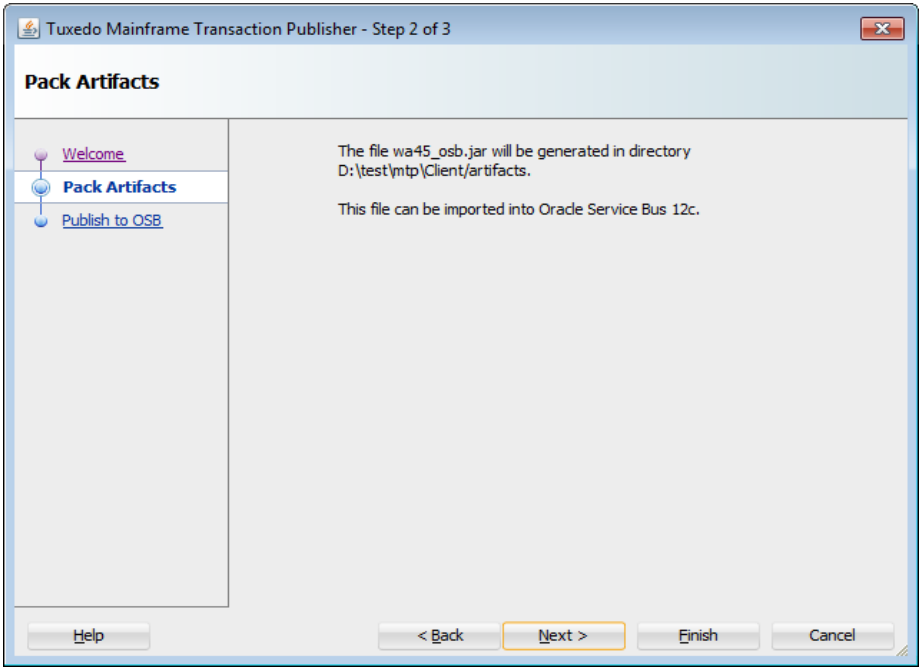
Tuxedo Mainframe Transaction Publisher is implemented through the UI hook. Users access this function by selecting the Tuxedo Mainframe Transaction Publisher menu item.

By selecting this function, a welcome wizard page will be displayed to do the following things.

1. Pack Artifacts
2. Publish to OSB

Pack Artifacts

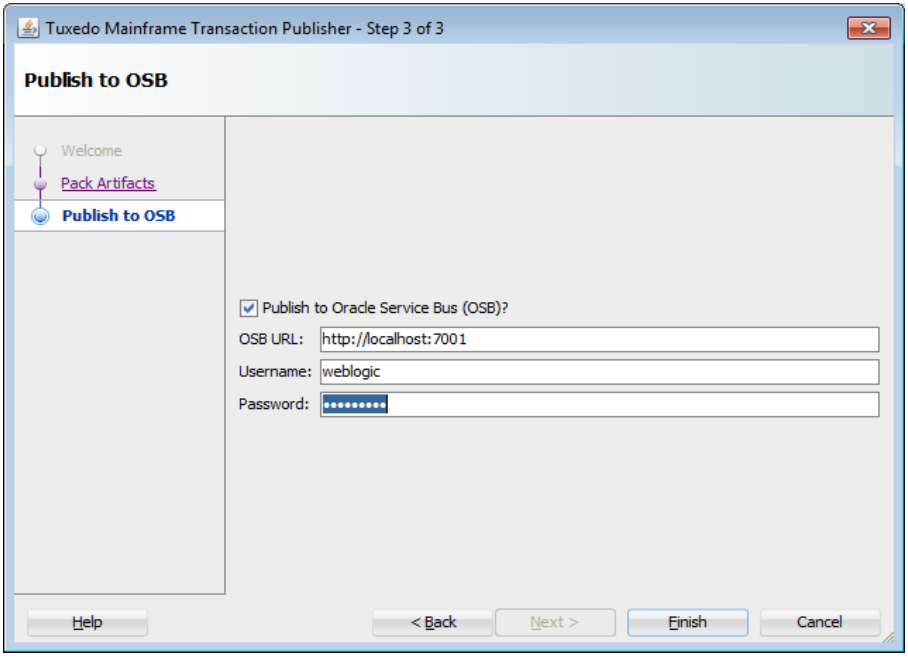
In this step, the artifacts generated by Tuxedo Mainframe Transaction Generator are packed. The following wizard page tells users the name of the packaged JAR file, and where it will be generated.



Publish to OSB

The following wizard page helps users to publish the generated artifacts to OSB. This Tuxedo Mainframe Transaction Publisher function allows users to specify the OSBs URL, administrator's name, and administrator's password.

Note: Tuxedo Mainframe Transaction Publisher allows users to manually install the OSB project by not selecting "Publish to Oracle Service Bus (OSB)?".



Installing/Uninstalling Tuxedo Mainframe Transaction Publisher

Prerequisite

To ensure successful installation of the Tuxedo Mainframe Transaction Publisher, a pristine JDeveloper should be used. Users should install the pristine JDeveloper at a new location; they should neither import any preference from other installations nor use JDeveloper to start from installer.

After installation, users use the following commands to start the JDeveloper.

- `cd $ORACLE_HOME`
- `jdeveloper/jdev/bin/jdev -clean -console`

Note: JDeveloper Studio is available for download from Oracle Technology Network.

Installing Tuxedo Mainframe Transaction Publisher

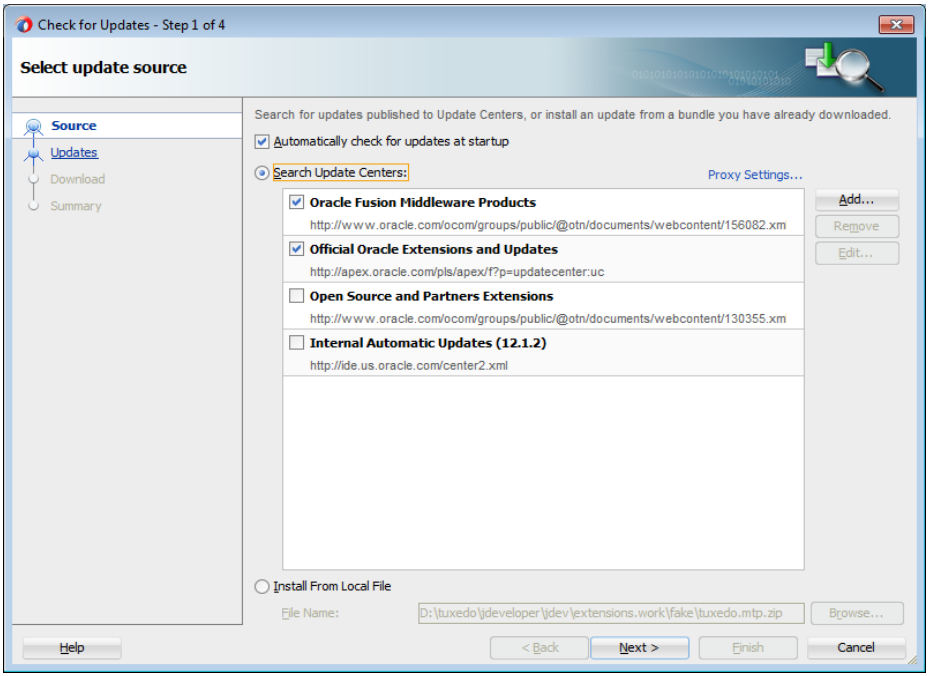
The Tuxedo Mainframe Transaction Publisher is distributed in a single zip file named "`tuxedo.mtp.update.<version>.zip`". Its current version is 12.1.2.0.

Do the following steps to complete the Tuxedo Mainframe Transaction Publisher installation.

1. Select "Install From Local File" and enter the zip file location in "File Name:" text field.
2. Click the "Next" button (and the "Summary" page shows up).
3. Click the "Finish" button to complete the installation.

After completing the installation, jar files will be installed in `MW_HOME/JDeveloper/jdev/extension/tuxedo` directory.

Note: The zip file is located in `$TUXDIR/udataobj`. To find out "`tuxedo.mtp.update.12.1.2.0.zip`", open the JDeveloper and click the "Help" menu item in the menu bar, and select "Check for Updates" from the drop down menu that is brought up.



Checking Installation Status

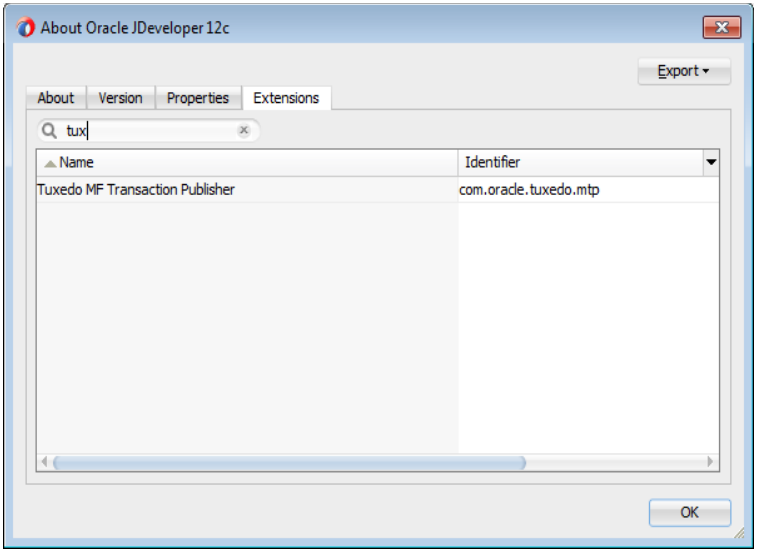
After the installation, when the updater asks to restart JDeveloper, choose not to. Then users go to the command line and re-enter `jdeveloper/jdev/bin/jdev -clean -console` to verify whether the installation is successful.

Users can check the installation status using any of the following ways.

- Using graphical user interface
- Using command lines

Using graphical user interface

Click "Help"- "About" - "Extension".



Using command lines

Listing 1 Using Command Lines to Check Installation Status

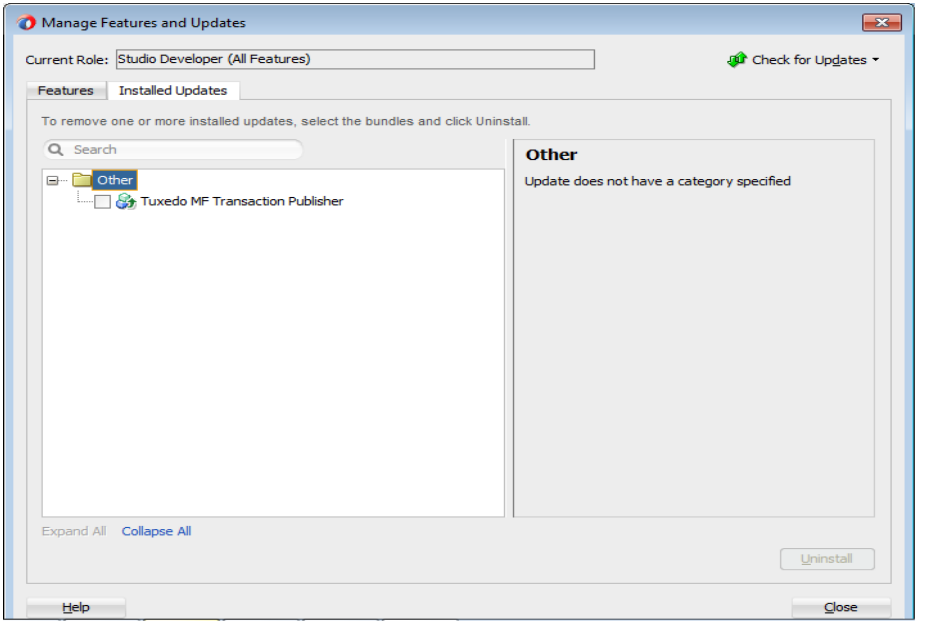
```
D:\oracle\jdeveloper\12.1.2_2>jdeveloper\jdev\bin\jdev -su -clean -console
```

```
osgi>

osgi> ss tuxedo

Framework is launched.

id      State       Bundle
927     RESOLVED    com.oracle.tuxedo.mtp_12.1.2
```

Uninstalling Tuxedo Mainframe Transaction Publisher

Do the following steps to uninstall the Tuxedo Mainframe Transaction Publisher from JDeveloper's menu bar.

1. Click the "Tools" menu item (and a drop down menu shows up).
2. Select the "Features" (and the "Manage Features and Updates" page shows up.
3. Select the "Installed Updates".
4. Select "Tuxedo MF Transaction Publisher".
5. Click "Uninstall" button to complete the uninstallation.



Installation Notes

Tuxedo Mainframe Transaction Publisher requires

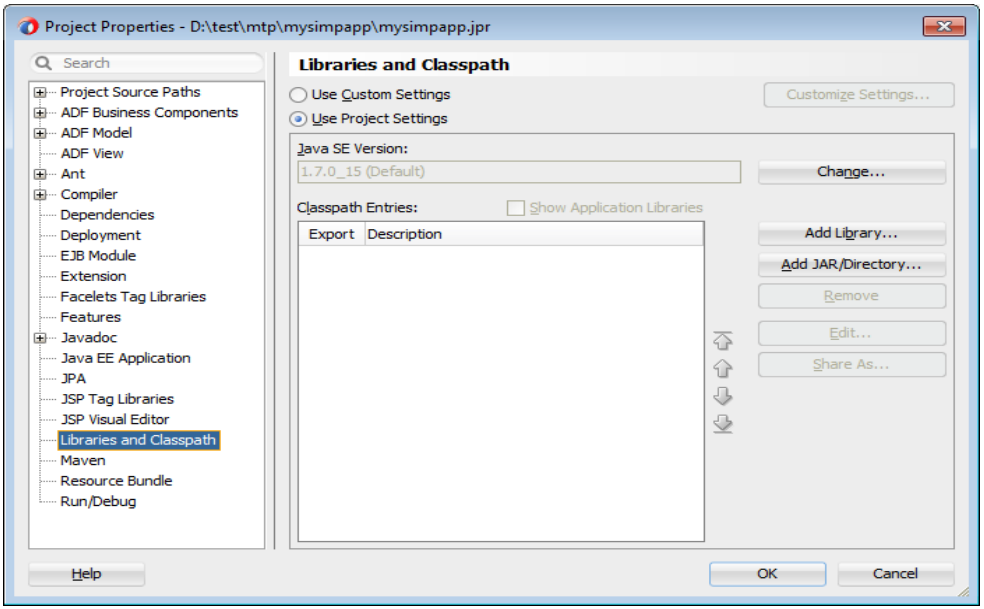
- Oracle JDeveloper 12.1.2 extension
- Oracle Service Bus (OSB) 11.1.1.7
- JDK 1.7 or above on both Oracle JDeveloper and Oracle Service Bus (OSB)

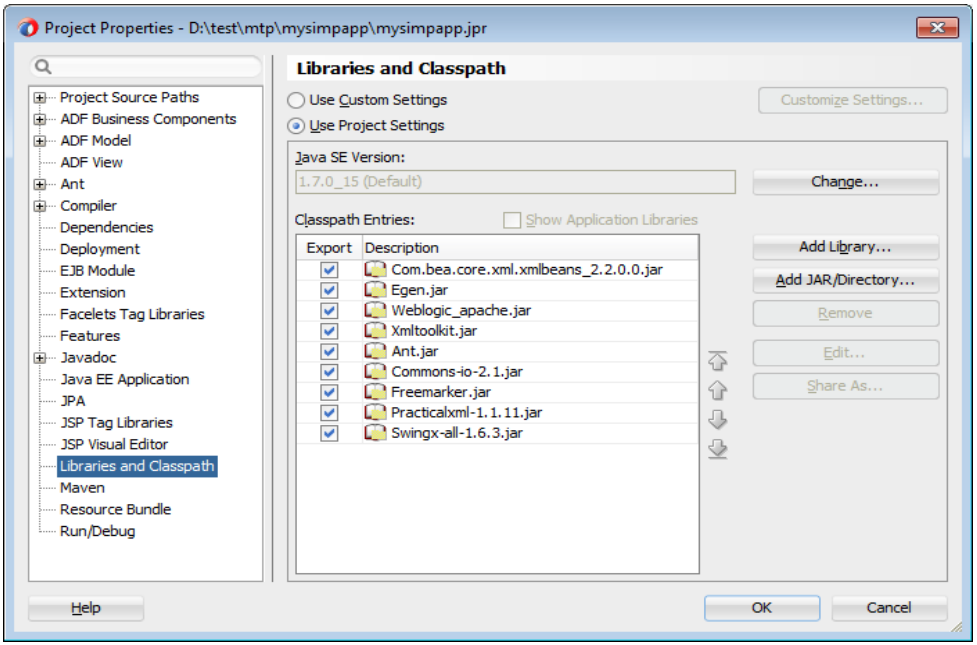
Note: When users install Tuxedo Mainframe Transaction Publisher on Oracle JDeveloper 12.1.2 extension, a matisse related exception will be reported. This exception has no impact on the use of Tuxedo Mainframe Transaction Publisher.

Setting up JDeveloper Project

Users must set up the "Library and Classpath" for every project before using Tuxedo Mainframe Transaction Publisher; otherwise, the compilation of the generated class fails.

To do the setup, right click the project to bring up context menu and select "Project Properties". Then select "Add JAR/Directory" and add the eGen libraries.


Project Properties - D:\test\mtp\mysimpapp\mysimpapp.jpr
Search
Project Source Paths
ADF Business Components
ADF Model
ADF View
Ant
Compiler
Dependencies
Deployment
EJB Module
Extension
Facelets Tag Libraries
Features
Javadoc
Java EE Application
JPA
JSP Tag Libraries
JSP Visual Editor
Libraries and Classpath
Maven
Resource Bundle
Run/Debug
Libraries and Classpath
Use Custom Settings
Use Project Settings
Customize Settings...
Java SE Version:
1.7.0_15 (Default)
Change...
Classpath Entries:
Show Application Libraries
Export
Description
Add Library...
Add JAR/Directory...
Remove
Edit...
Share As...
Help
OK
Cancel


Project Properties - D:\test\mtp\mysimpapp\mysimpapp.jpr
Project Source Paths
ADF Business Components
ADF Model
ADF View
Ant
Compiler
Dependencies
Deployment
EJB Module
Extension
Facelets Tag Libraries
Features
Javadoc
Java EE Application
JPA
JSP Tag Libraries
JSP Visual Editor
Libraries and Classpath
Maven
Resource Bundle
Run/Debug
Libraries and Classpath
Use Custom Settings
Use Project Settings
Customize Settings...
Java SE Version:
1.7.0_15 (Default)
Change...
Classpath Entries:
Show Application Libraries
Export
Description
Com.bea.core.xml.xmlbeans_2.2.0.0.jar
Egen.jar
Weblogic_apache.jar
Xmltoolkit.jar
Ant.jar
Commons-io-2.1.jar
Freemarker.jar
Practicalxml-1.1.11.jar
Swingx-all-1.6.3.jar
Add Library...
Add JAR/Directory...
Remove
Edit...
Share As...
Help
OK
Cancel

Setting up Oracle Service Bus (OSB)

Installing EGen Libraries for OSB

It is required for users to add eGen libraries to OSB's classpath by doing the following steps.

1. Create or use an existing Oracle Service Bus Domain.
2. Edit `<domain_path>/bin/setDomainEnv.sh` and eGen libraries to the classpath.
3. Restart OSB to reflect these changes in the classpath.

The eGen libraries can be extracted from the updated zip file.

Users should add the followings to `setDomainEnv.sh`.

Listing 2 Adding Information to setDomainEnv.sh

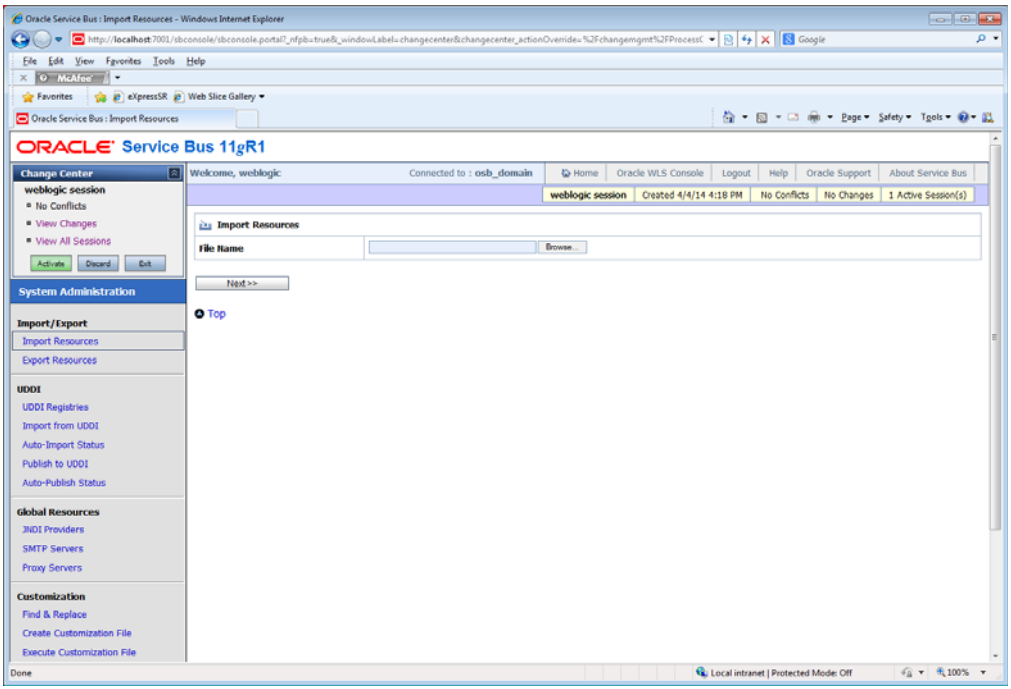
```
#
# EGen Classpath for MTP
#
BASE_EGEN_LIBS_PATH=<location of the libraries>
EGEN_CLASSPATH=${BASE_EGEN_LIBS_PATH}/com.bea.core.xml.xmlbeans_2.2.0.0.jar${CLASSPATHSEP}${BASE_EGEN_LIBS_PATH}/weblogic_apache.jar${CLASSPATHSEP}${BASE_EGEN_LIBS_PATH}/xmltoolkit.jar${CLASSPATHSEP}${BASE_EGEN_LIBS_PATH}/egen.jar
CLASSPATH="${CLASSPATH}${CLASSPATHSEP}${EGEN_CLASSPATH}"
export CLASSPATH
```

Importing Shared Resources to OSB

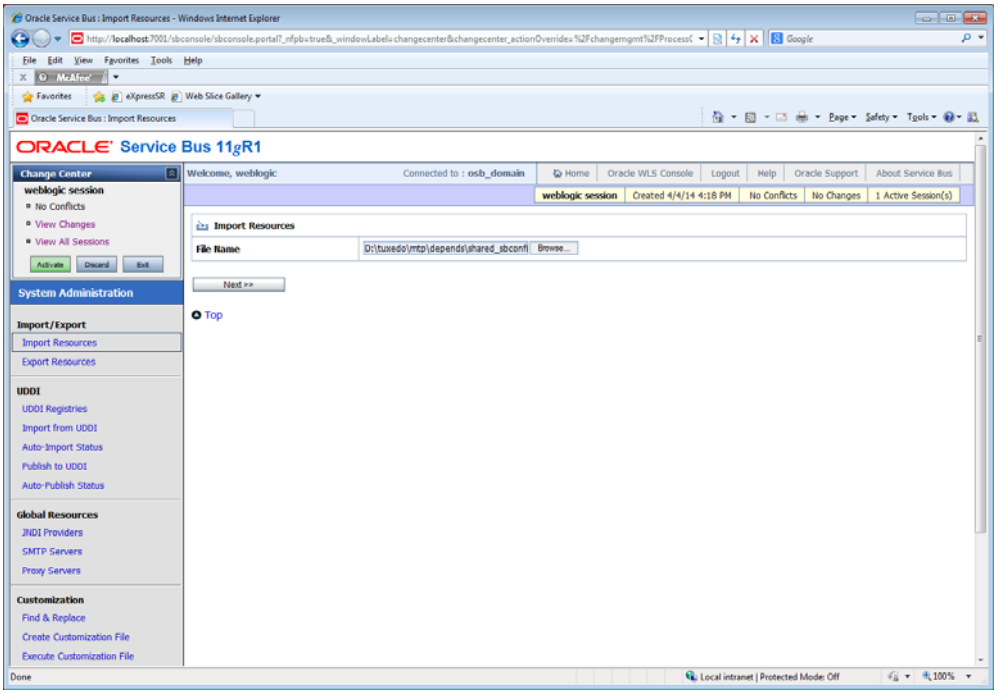
An OSB project with some shared resources is used by Tuxedo Mainframe Transaction Publisher generated OSB resources. The file with complete OSB project is in `$TUXDIR/udataobj/mtp_shared_sbconfig.jar`.

1. Use OSB's console to import this JAR.

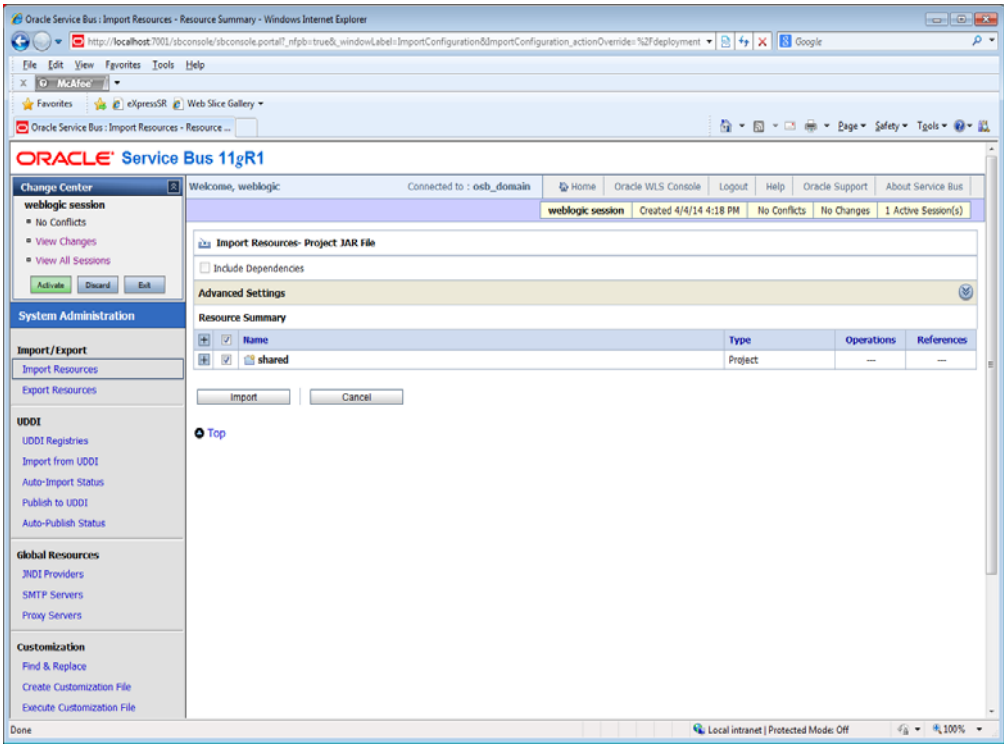
`System Administration > Import Resources`



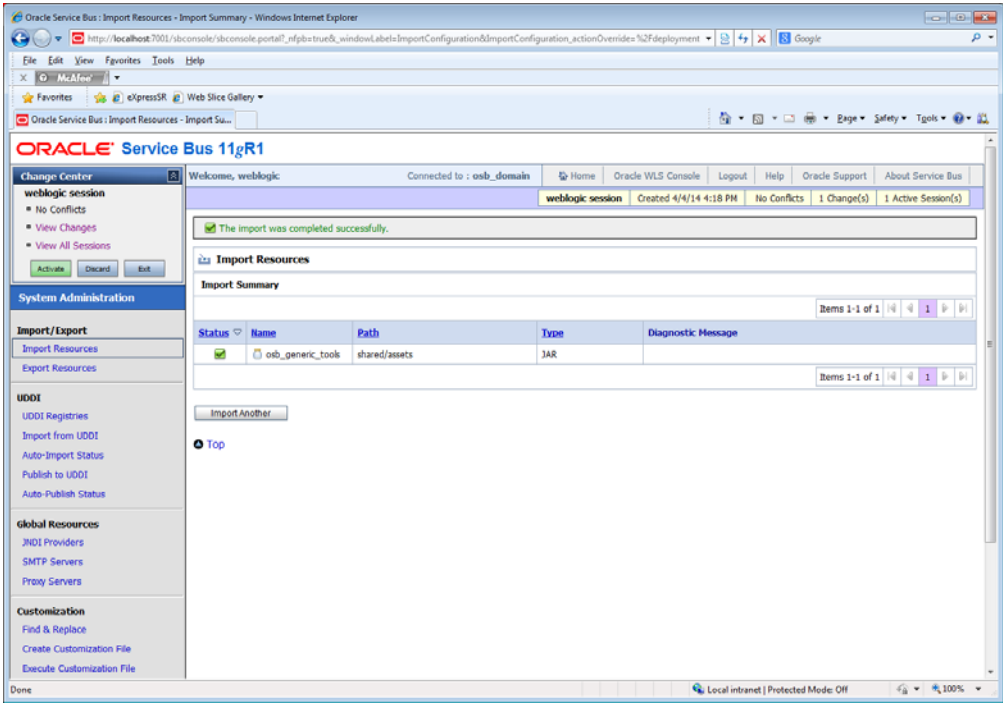
2. Enter the `mtp_shared_sbconfig.jar` location.



3. Click "Next>>" button.



4. Select "Import".



5. Click "Activate" button.
6. Click "Submit" button.
7. Check for any error or conflict and resolve them.