Oracle® Java CAPS XSLT Service Engine Tutorial

Part No: 821–2616
March 2011

120126@25097

Contents

XSLT Designer: Simple Transformation Tutorial 5
Overview 5
Configuring the Tutorial Environment 5
Creating the XSLT Module Project 6
▼ To Create a New XSLT Module Project 6
Creating XML Schemas 7
▼ To Create the XML Schema for the Incoming Message 7
▼ To Create the XML Schema for the Outgoing Message 9
Creating a WSDL File 10
▼ To Create a WSDL File 10
Creating an XSLT Service 11
▼ To Create an XSLT Service 11
▼ To Populate the XSL Stylesheet 12
Creating and Deploying the Composite Application 13
▼ To Create a Composite Application Project 13
▼ To Add a JBI Module 14
▼ To Deploy the HelloXSLTCAP Composite Application 14
Performing a Test Run of the XSL Transformation Service 15
▼ To Create a Test Case 15
▼ To Test the Application: 17

XSLT Designer: Simple Transformation Tutorial

The list below comprises the subjects covered in this topic:

- "Overview" on page 5
- "Configuring the Tutorial Environment" on page 5
- "Creating the XSLT Module Project" on page 6
- "Creating XML Schemas" on page 7
- "Creating a WSDL File" on page 10
- "Creating and Deploying the Composite Application" on page 13
- "Performing a Test Run of the XSL Transformation Service" on page 15

Overview

In this tutorial you become acquainted with the XSLT Service Designer, which is part of the Oracle Java Composite Application Suite (Java CAPS). The XSLT Designer is used to develop, deploy and test XSL Transformation Services. An XSL Transformation Service acts as a web service. It receives messages from a client, transforms them, and either sends the messages back to the originator or forwards them to another web service.

In this tutorial you will create a simple XSL Transformation Service that receives a message, transforms it, and sends it back to the calling web service.

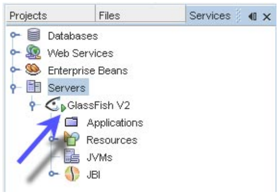
Configuring the Tutorial Environment

Perform the following steps to confirm that GlassFish V2 Application Server is installed and that the JBI runtime contains the XSLT Service Engine and Transform Shared Library required for this tutorial:

1. Open the Services window.
2. Expand the `Servers` node.

3. Right-click the `GlassFish V2` node and choose Start form the popup menu.

   If the Start option is not available and there is a green "badge" next to the `GlassFish V2` node, that means the server is already running.



4. After the server is started, expand the GlassFish V2 > JBI node. Then expand the Shared Libraries node to verify that sun-wsdl-ext-library is installed.

Creating the XSLT Module Project

An XSL Transformation Service is created within an XSLT Module project.

▼ To Create a New XSLT Module Project

1 **From the IDE's main menu, choose File > New Project.**

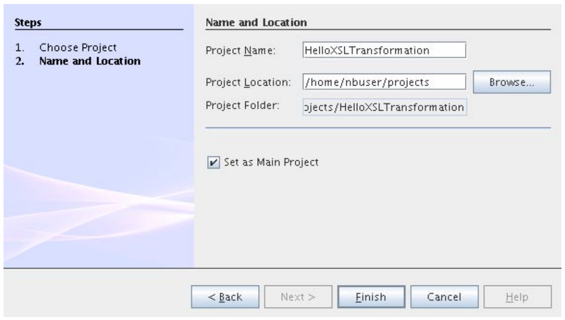
2 **Under Categories select SOA.**

3 **Under Projects, select XSLT Module.**

4 **Click Next.**

5 **In the Project Name field, type HelloXSLTransformation.**

6 **Modify the project location, or accept the default.**



Steps

1. Choose Project
2. Name and Location

Name and Location

Project Name: HelloXSLTransformation

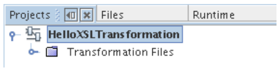
Project Location: /home/nbuser/projects

Project Folder: ojects/HelloXSLTransformation

[✓] Set as Main Project

7 **Click Finish.**

8 **The Projects window now contains the HelloXSLTransformation project node.**



9 **Expand the `HelloXSLTransformation` > `Transformation Files` node. Note that the `transformmap.xml` file was created, which is a custom configuration file used to define transformation processes.**

In the next steps you create two XML Schema (`.xsd`) files, a web service description (`.wsdl`) file and an XSL stylesheet (`.xsl`) file. To run an XSL Transformation Service, you need at least one XML Schema, one WSDL file and one XSL stylesheet. For the purpose of this tutorial, you create two XML Schemas.

Creating XML Schemas

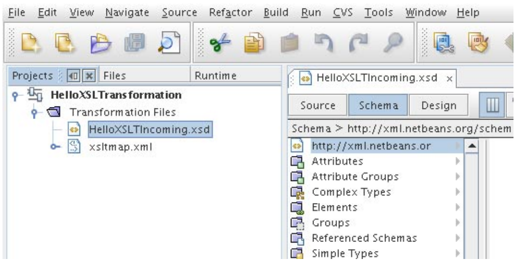
In this step you are going to create two XML Schema files: `HelloXSLTIncoming.xsd` as a basis for the incoming message, and `HelloXSLTOutgoing.xsd` as a basis for the outgoing message.

▼ To Create the XML Schema for the Incoming Message

1 **In the Projects window, right-click the HelloXSLTransformation > Transformation Files node and choose New > XML Schema.**

2 **In the File Name field, type HelloXSLTIncoming.**

3. **Click Finish. A new node—HelloXSLTIncoming.xsd—appears under the Transformation Files node in your HelloXSLTransformation project and the new Schema opens in the XML Schema Editor.**

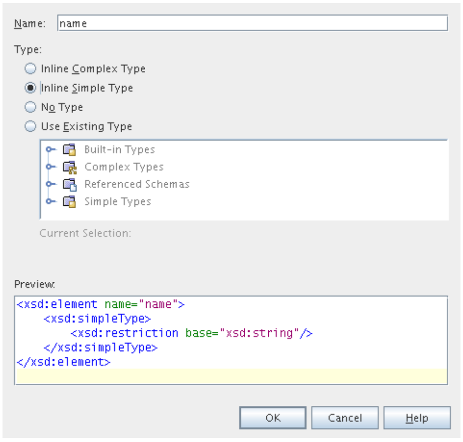


4. **In the first column of the Schema view, right-click Elements and choose Add Element from the popup menu. The Add Element dialog box opens.**

5. **In the Name field, type `name`.**

6. **Under Type, select the Use Existing Type radio button.**

7. **Expand the Built-in Types node and select string, and click OK.**


Name: name

Type:
- Inline Complex Type
- ● Inline Simple Type
- No Type
- Use Existing Type

Preview:

```
<xsd:element name="name">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:element>
```

8. **To view the source of the Schema you created, click the Source button on the XML Schema Editor toolbar. You should see the following code:**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/HelloXSLTIncoming"
            xmlns:tns="http://xml.netbeans.org/schema/HelloXSLTIncoming"
            elementFormDefault="qualified">
    <xsd:element name="name" type="xsd:string"></xsd:element>
</xsd:schema>
```

▼ To Create the XML Schema for the Outgoing Message

1. **In the Projects window, right-click the HelloXSLTransformation > Transformation Files node and choose New > XML Schema.**

2. **In the File Name field, type HelloXSLTOutgoing.**

3. **Click Finish. A new node—HelloXSLTOutgoing.xsd —appears under the Transformation Files node in your HelloXSLTransformation project and the new Schema opens in the XML Schema Editor.**

4. **In the first column of the Schema view, right-click Elements and choose Add Element from the popup menu. The Element dialog box opens.**

5. **In the Name field, type `greeting`.**

6. **Under Type, select the Use Existing Type radio button.**

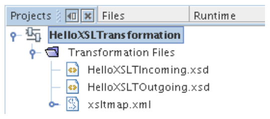
7. **Expand the Built-in Types node, select string and click OK.**

8. **To view the source of the Schema you created, click the Source button on the XML Schema Editor toolbar. You should see the following code:**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/HelloXSLTOutgoing"
            xmlns:tns="http://xml.netbeans.org/schema/HelloXSLTOutgoing"
            elementFormDefault="qualified">
    <xsd:element name="greeting" type="xsd:string"></xsd:element>
</xsd:schema>
```

9. **Click the Save All button on the toolbar.**

   You should see two Schema files listed under the `Transformation Files` node in your `HelloXSLTransformation` project.



Creating a WSDL File

In this step you create a web service description file that defines the web interface of our XSLT Service.

▼ To Create a WSDL File

1. **In the Projects window, right-click the HelloXSLTransformation > Transformation Files node and choose New > WSDL Document.**

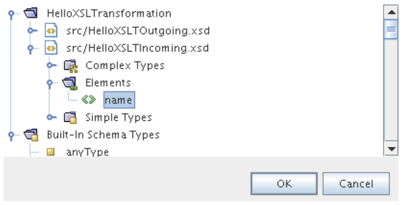
2. **In the File Name field, type `HelloXSLTWSDL`.**

3. **Select Concrete WSDL Document as the WSDL type.**

4. **Select SOAP from the Binding drop-down list, and Document Literal from the Type drop-down list, then click Next.**

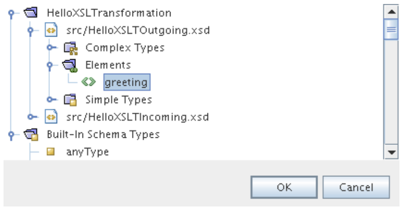
5. **The Abstract Configuration window will open. Under Input, in the Element Or Type column, click the ellipsis button. The Select Element Or Type dialog box opens.**

6. **Scroll up and select By File > HelloXSLTransformation > src/HelloXSLTIncoming.xsd > Elements > name and click OK.**



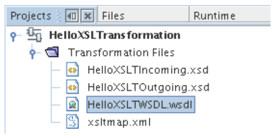
7. **Under Output, in the Element Or Type column, click the ellipsis button. The Select Element Or Type dialog box opens.**

8. **Select By File > HelloXSLTransformation > src/HelloXSLTOutgoing.xsd > Elements > greeting and click OK.**



9 **Click Next.**

10 **On the Concrete Configuration page review the values and click Finish. You should see the `HelloXSLTWSDL.wsdl` file listed under the Transformation Files node in your HelloXSLTransformation project.**



Note – Creating and editing WSDL files is not covered in this tutorial. For more information about the WSDL Editor, see *Oracle Java CAPS WSDL Editor User's Guide*.

Creating an XSLT Service

In this step, you will create a simple request-reply XSLT service, which is a web service that receives input messages, transforms them in accordance with an XSL stylesheet, and sends them back.

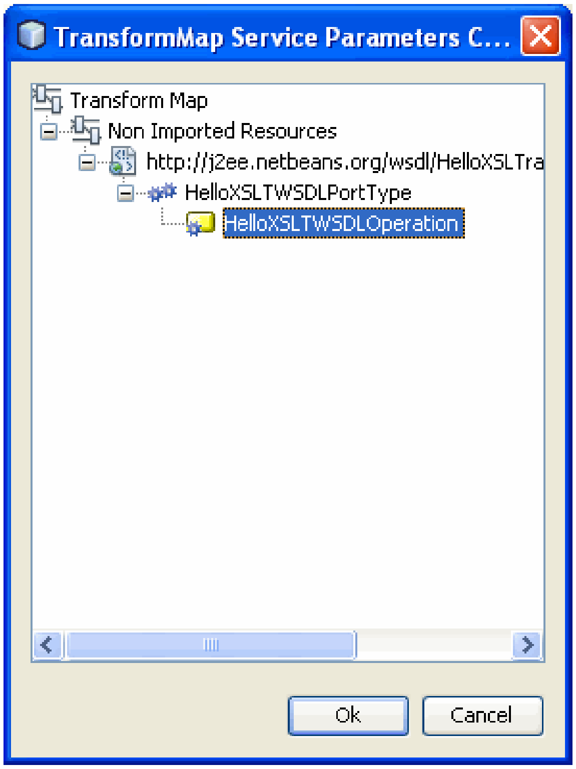
▼ To Create an XSLT Service

1 **In the Projects window, right-click the HelloXSLTransformation > Transformation Files node and choose New > XSLT Service.**

2 **In the Service Type window, ensure that the Request-Reply Service radio button is selected and click Next.**

3 **In the next step, enter `HelloXSLTService` as the service name and click Choose to select Operation for the service.**

4 **In the TransformMap Service Parameters window, expand the upper node with the target namespace until you see the operation, select HelloXSLTWSDLOperation and click Next.**


TransformMap Service Parameters C...

Transform Map
Non Imported Resources
http://j2ee.netbeans.org/wsdl/HelloXSLTra
HelloXSLTWSDLPortType
HelloXSLTWSDLOperation

5 **Click Finish. The HelloXSLTService.xsl file opens in the XSLT Editor, where you can only view and edit the source code.**

Now you need to populate the XSL stylesheet with the transformation rules. The XSL stylesheet defines how to transform the input XML document. In this version of the XSLT Designer, you can only edit the source code in the Source view.

▼ To Populate the XSL Stylesheet

1 **Open the `HelloXSLTService.xsl` file and type in the following code:**

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.

xmlns:ns1="http://xml.netbeans.org/schema/HelloXSLTIncoming

xmlns:ns="http://xml.netbeans.org/schema/HelloXSLTOutgoing"
```

```
<xsl:template match="/">

<xsl:element name="ns:greeting">

<xsl:value-of select="concat(&apos;Hello &apos;, /ns1:name)"/>

</xsl:element>

</xsl:template>

</xsl:stylesheet>
```

2 **Press the Validate File button on the toolbar to ensure that the code has no errors.**

3 **Right-click the HelloXSLTransformation node and choose Clean and Build from the drop-down menu to build the project. You should see the `BUILD SUCCESSFUL` message after building the project completes.**

Creating and Deploying the Composite Application

An XSLT project is not directly deployable. You must first add an XSLT project as a JBI module to a Composite Application project before you can deploy the Composite Application project. Deploying the project makes the service assembly available to the application server as a JBI service unit. After deployment, you will be able to perform a test run of your XSLT service.

▼ To Create a Composite Application Project

1 **Choose File > New Project from the main menu.**

2 **Under Categories, select SOA.**

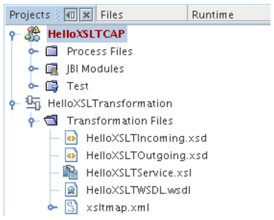
3 **Under Projects, select Composite Application. Click Next.**

4 **In the Project Name field, type `HelloXSLTCAP`.**

5 **Specify a project location or accept the default.**

6 **Click Finish.**

7 **The Projects window now contains the HelloXSLTCAP project node.**

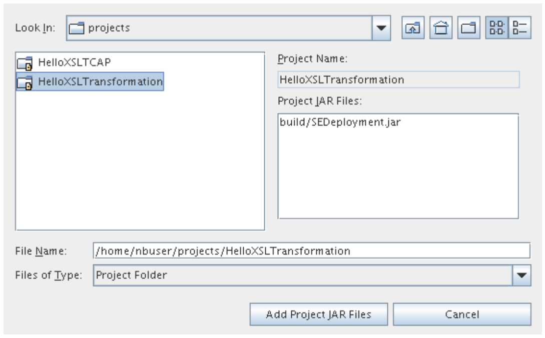


HelloXSLTCAP
Process Files
JBI Modules
Test
HelloXSLTransformation
Transformation Files
HelloXSLTIncoming.xsd
HelloXSLTOutgoing.xsd
HelloXSLTService.xsl
HelloXSLTWSDL.wsdl
xsltmap.xml

To Add a JBI Module

1 **Right-click the HelloXSLTCAP node and choose Add JBI Module from the popup menu.**

2 **Select the HelloXSLTransformation project and click Add Project Jar Files.**



Look In: projects

HelloXSLTCAP
HelloXSLTransformation

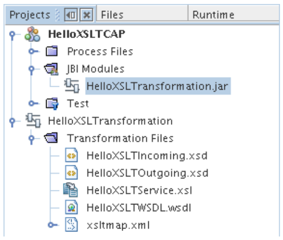
Project Name: HelloXSLTransformation

Project JAR Files: build/SEDeployment.jar

File Name: /home/nbuser/projects/HelloXSLTransformation

Files of Type: Project Folder

3 **To verify that the JBI module has been added, expand HelloXSLTCAP > JBI Modules.**



HelloXSLTCAP
Process Files
JBI Modules
HelloXSLTransformation.jar
Test
HelloXSLTransformation
Transformation Files
HelloXSLTIncoming.xsd
HelloXSLTOutgoing.xsd
HelloXSLTService.xsl
HelloXSLTWSDL.wsdl
xsltmap.xml

To Deploy the HelloXSLTCAP Composite Application

1 **In the Projects window, right-click the HelloXSLTCAP node and choose Deploy Project from the popup menu.**

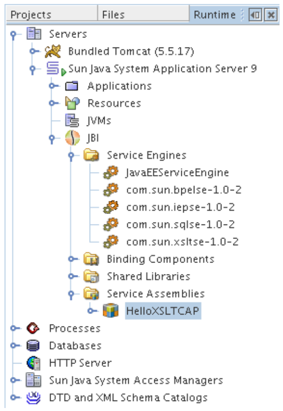
This operation might take a while because the it starts the GlassFish Application Server.

Note: If the Warning - Select Server dialog box appears, select the Application Server and click OK.

2 **In the Output window that opens in the lower part of the IDE, watch for the `BUILD SUCCESSFUL` message.**

3 **To verify that the project has been deployed, expand GlassFish V2 > JBI > Service Assemblies in the Runtime window.**

You should see the HelloXSLTCAP node.



Performing a Test Run of the XSL Transformation Service

Testing an XSL Transformation Service means sending a message that the Service is expecting and receiving, in this case, a reply message.

Before we can perform the testing, we must create a test case.

▼ To Create a Test Case

1 **In the Projects window, expand the HelloXSLTCAP node and right-click the Test node.**

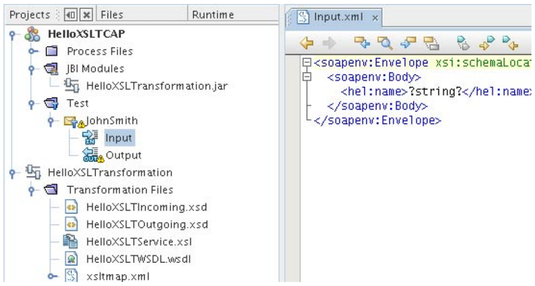
2 **From the popup menu, select New Test Case.**

3 **In the Test Case Name field, type `JohnSmith`. Click Next.**

4 **Under Select the WSDL Document, expand HelloXSLTransformation - XSLT Process Files and select `HelloXSLTWSDL.wsdl`. Click Next.**

5 **Under Select the Operation to Test, expand HelloXSLTWSDLBinding and select HelloXSLTWSDLOperation. Click Finish.**

6 The JohnSmith node appears under HelloXSLTCAP > Test and the input message file—`Input.xml`—opens in the editor.



```
<soapenv:Envelope xsi:schemaLoca
  <soapenv:Body>
    <hel:name>?string?</hel:name
  </soapenv:Body>
</soapenv:Envelope>
```

7 In the `Input.xml` file, modify the

```
<hel:name>?string?</hel:name>
```

line to

```
<hel:name>John Smith</hel:name>
```

The `Input.xml` file should be:

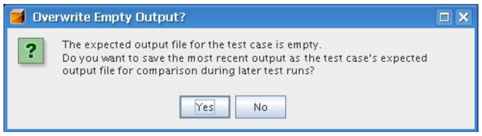
```
 <soapenv:Envelope xsi:schemaLocation=
"http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/
soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:hel="http://xml.netbeans.org/schema/HelloXSLTIncoming">

        <soapenv:Body>

                <hel:name>John Smith</hel:name>

        </soapenv:Body>

</soapenv:Envelope>
```

8 Click the Save All button on the toolbar.

The `Output` node under the test case node refers to the expected reply message that is used for comparison with the actual reply messages. Before we run the test for the first time, the `Output.xml` file is empty. The first test run will populate `Output.xml` with the real output. Subsequent test runs will compare the real output against the content of `Output.xml`

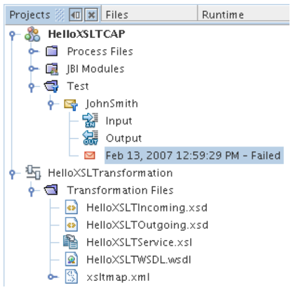
▼ To Test the Application:

1 **Right-click the JohnSmith node and select Run. Notice that the test fails and the following dialog box appears:**


Overwrite Empty Output?

The expected output file for the test case is empty.
Do you want to save the most recent output as the test case's expected output file for comparison during later test runs?

2 **Click Yes. Notice that the failed test node appears below the Output node.**



3 **Double-click the failed test node to see the message that the XSL Transformation Service sent back:**

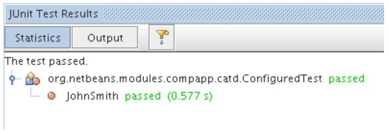
```
<?xml version="1.0" encoding="UTF-8"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://xml.netbeans.org/
schema/HelloXSLTOutgoing">

        <SOAP-ENV:Header/>

        <SOAP-ENV:Body>

                <ns:greeting xmlns:ns="http://xml.netbeans.org/schema/
HelloXSLTOutgoing">Hello John Smith</ns:greeting>

        </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Notice the line

```
<ns:greeting xmlns:ns="http://xml.netbeans.org/schema/HelloXSLTOutgoing">Hello John Smith</ns:greeting>
```

The XSL Transformation Service received the name, concatenated it with the string 'Hello' and sent the reply message.

4 Run the test again. The test is marked as passed.



```
The test passed.
org.netbeans.modules.compapp.catd.ConfiguredTest passed
    JohnSmith passed (0.577 s)
```

You have successfully created, deployed and tested an XSL Transformation Service.

Now that you have successfully created the Request-Reply XSL Transformation Service, continue with the Service Bridge type.