XSLT Designer Quick Start Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–4051
December 2008

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

081223@21808

Contents

Understanding the XSLT Designer

The list below comprises the subjects covered in this topic:

Overview

In this tutorial you become acquainted with the XSLT Designer included in NetBeans IDE 6.1. 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 an external client, transforms the messages in accordance with an XSL stylesheet, and either sends the messages back to the originator or forwards them to another web service.

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

Prerequisites

This tutorial assumes that you have some basic knowledge of, or programming experience with, the NetBeans IDE.

System Requirements

This tutorial assumes that your system meets the requirements specified in the *System Requirements* section of the Netbeans IDE 6.1 Release Notes.

Software Needed for the Tutorial

Before you begin, download and install the following software on your computer: NetBeans IDE 6.1 (`http://www.netbeans.org/downloads/index.html`). Click the **Download** button under the **All** column. This option includes all the features and servers required for this tutorial.

Configuring the Tutorial Environment

This tutorial requires that the GlassFish V2 Application Server, which includes the JBI runtime, has been installed with NetBeans IDE 6.1. Perform the following steps to confirm that GlassFish V2 Application Server is installed with NetBeans IDE 6.1 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 pop-up menu.

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

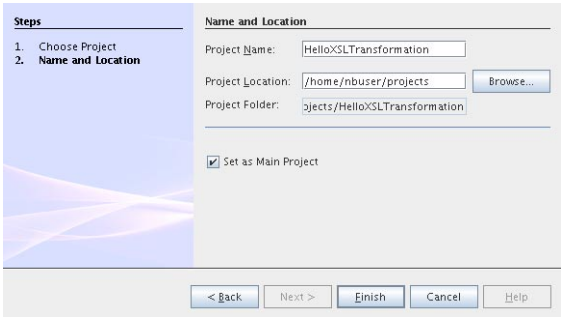
4. After the server is started, expand the `GlassFish V2` > `JBI` node. Then expand the `Shared Libraries` node to verify that `sun-dwdl-ext-library` is installed.

Creating the XSLT Module Project

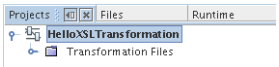
An XSL Transformation Service is created within an XSLT Module project.

▼ 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.**



7. **Click Finish.**
8. **The Projects window now contains the** `HelloXSLTransformation`**a project node.**



Next 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

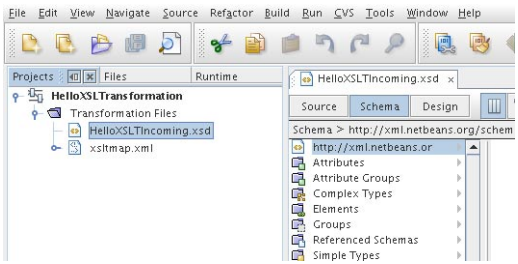
You will create two XML Schemas: `HelloXSLTIncoming.xsd` and `HelloXSLTOutgoing.xsd`. You use the former as the basis for the incoming message and the latter as the basis for the outgoing message.

▼ 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.**



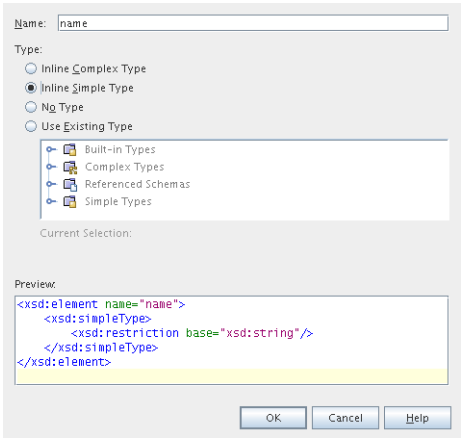
4 **In the first column of the Schema view, right-click Elements and choose Add Element from the pop-up menu. The Add Element dialog box opens.**

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`**.**

8 **Click OK.**



9 **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 pop-up 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 and select `string`.**

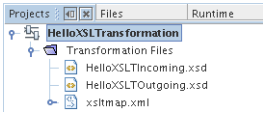
8 **Click OK.**

9 **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>
```

10 **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

Next you create a web service description file defining the web interface of our XSL Transformation Service.

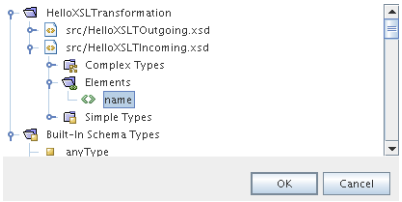
▼ To create a WSDL file:

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`, then click Next.**

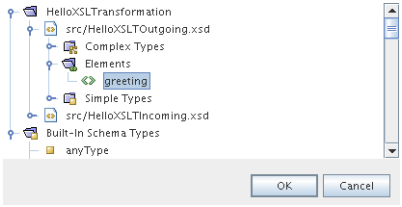
3 **Under Input, in the Element Or Type column, click the ellipsis (...) button. The Select Element Or Type dialog box opens.**

4 **Select `By File` > `HelloXSLTransformation` > `src/HelloXSLTIncoming.xsd` > `Elements` > `name` and click OK.**



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

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

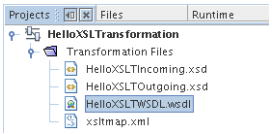


7 **Click Next.**

8 **In the Binding Type Field, select SOAP.**

9 **Under Binding Subtype, select Document Literal, then click Finish.**

You should see the `HelloXSLTWSDL.wsdl` file listed under the `Transformation Files` node in your `HelloXSLTransformation` project.



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, thus allowing its service units to be run.

▼ To create a Composite Application:

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

2 **Under Categories, select** `SOA`**.**

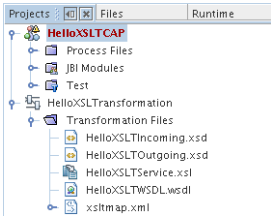
3 **Under Projects, select** `Composite Application`**. Click Next.**

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

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

6 **Click Finish.**

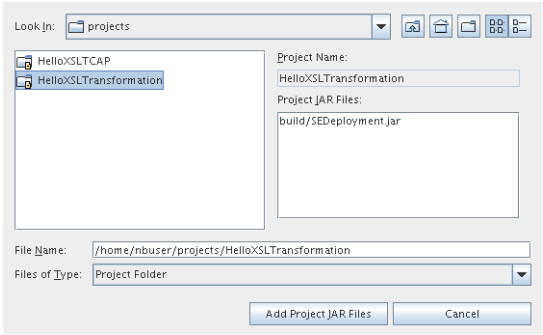
7 **The Projects window now contains the** `HelloXSLTCAP` **project node.**



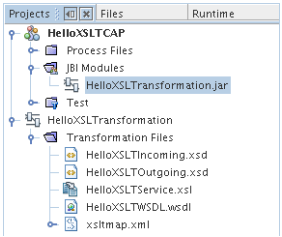
▼ To add a JBI module:

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

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



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



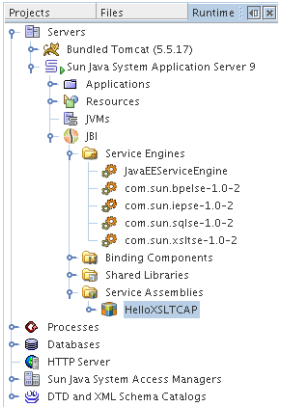
▼ To deploy the HelloXSLTCAP Composite Application:

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

Note: If the Warning - Select Server dialog box appears, select Sun Java System Application Server 9 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** `Sun Java System Application Server 9` > `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.**

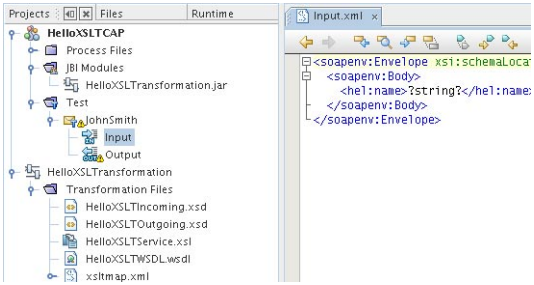
2 **From the pop-up menu, select New Test Case.**

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.**



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

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

line to

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

The `Input.xml` file should be:

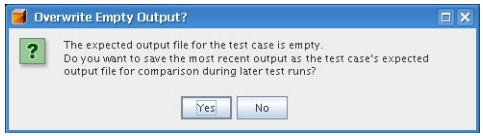
```
 <soapenv:Envelope xsi:schemaLocation=
"http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/
soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:hel="http://xml.netbeans.org/schema/HelloXSLTIncoming">

        <soapenv:Body>

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

        </soapenv:Body>

</soapenv:Envelope>
```

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

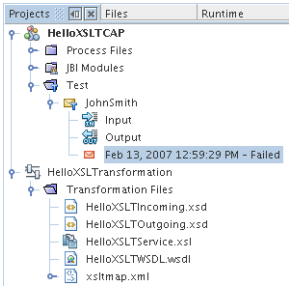
The `Output` node under the test case node refers to the expected reply message that is used for comparison with the actual reply messages. Before we run the test for the first time, the `Output.xml` file is empty. We will populate it with the content of the reply message (provided that it is what we expect).

▼ To run the test:

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



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:**

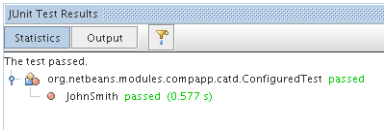
```
<?xml version="1.0" encoding="UTF-8"?>

<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.**



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.