

# Oracle® Java CAPS XSLT Editor User's Guide

Copyright © 2009, 2011, Oracle and/or its affiliates. All rights reserved.

#### **License Restrictions Warranty/Consequential Damages Disclaimer**

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

#### **Warranty Disclaimer**

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

#### **Restricted Rights Notice**

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

##### **U.S. GOVERNMENT RIGHTS**

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

#### **Hazardous Applications Notice**

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

#### **Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group in the United States and other countries.

#### **Third Party Content, Products, and Services Disclaimer**

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Contents

---

<b>Using the XSLT Editor</b> .....	5
About the XSLT Service Engine .....	5
XSLT Service Engine Features .....	6
Request-Reply (requestReplyService) .....	6
Invoke-Send (filterOneWay) .....	6
Request-Invoke-Reply Chain (filterRequestReply) .....	6
Runtime Properties .....	7
Starting the Application Server .....	7
Viewing Service Engine Properties .....	7
Runtime Property Descriptions .....	8
Configuring the XSLT Service Engine .....	9
Use Case 1 .....	9
Use Case 2 .....	9
Configuring the XSLT Service Engine .....	10
transformJBI attribute .....	10
messageType attribute .....	11
XSLT Service Engine Component Limitations — Running the JBI Runtime Separately ....	11



# Using the XSLT Editor

---

The topics listed below provide an overview of the XSLT Service Engine, the XSLT Editor, and its relationship with the JBI runtime environment.

- [“About the XSLT Service Engine” on page 5](#)
- [“XSLT Service Engine Features” on page 6](#)
- [“Runtime Properties” on page 7](#)
- [“Configuring the XSLT Service Engine” on page 9](#)

## About the XSLT Service Engine

The XSLT Service Engine is a Java-based transformation engine that is used to convert XML documents from one data format to another. The XSLT Service Engine makes it easier for users to configure and expose XSL style sheets as web services. Using the XSLT Service Engine requires no special knowledge of XSL, but rather allows any XSL style sheet to be deployed as a JBI service unit.

The XSLT Service Engine is not solely responsible for performing transformations. XSL style sheets implement a web service operation (as normally defined in a WSDL). When deployed as JBI service units, these service units correspond to a service endpoint. Each endpoint is activated when the XSLT service unit is deployed. In a sense, the XSLT Service Engine is a container of XSL style sheets, each of which represents a service endpoint in the JBI environment.

The following steps highlight the life cycle of a typical message using the XSLT Service Engine:

1. The XSLT service unit is configured with service endpoint information.
2. The service unit is deployed, along with the XSL style sheet, to the JBI environment.
3. The XSLT Service Engine compiles the style sheet.
4. A message arrives and the XSLT Service Engine searches for the service endpoint responsible for handling the message.

5. The message is transformed using the service endpoint's XSL style sheet.
6. A response is sent back via the Normalized Message Router (NMR).

## XSLT Service Engine Features

The XSLT Service Engine supports the following use cases:

- Request-Reply  
(`requestReplyService`)
- Invoke-Send  
(`filterOneWay`)
- Request-Invoke-Reply Chain  
(`filterRequestReply`)

### Request-Reply ( `requestReplyService` )

Request-Reply is a standard request-reply scenario. An XML message request is transformed and the result is sent back to the original consumer of the XSLT endpoint. The sequence of events includes:

1. XML message in
2. XSL transformation of message
3. Transformed result out

### Invoke-Send ( `filterOneWay` )

Invoke-Send is a standard one-way invocation. An XML message request is transformed, and the result is sent (not to the original consumer but rather) to another endpoint in the JBI environment. The sequence of events includes:

1. XML message in
2. XSL transformation of message
3. Transformed result out to third party

### Request-Invoke-Reply Chain ( `filterRequestReply` )

The Request-Invoke-Reply chain is a representation of the adapter pattern; it applies two separate XSL transformations. This scenario is useful when there are two existing web services which must be integrated even though they have incompatible input and output.

The first existing service acts as a consumer to the XSLT endpoint, sending a request. This message is transformed to match the input of the second service.

The second service is then invoked in an asynchronous manner. When the response from the second service arrives via the NMR, it is transformed to match the expected reply to the first (consuming) service. The sequence of events includes:

1. XML message in
2. XSL transformation of message using first XSL style sheet
3. Invoke service, sending transformed message as input
4. Receive response from invoked service endpoint
5. XSL transformation of response using second XSL style sheet
6. Reply to original sender with transformed third party response

## Runtime Properties

Changes to the XSLT Service Engine runtime properties are made from within NetBeans.

### Starting the Application Server

Configuration of XSLT Service Engine runtime properties requires first starting the GlassFish Application Server in the NetBeans window.

#### ▼ To Start the GlassFish Application Server

- 1 Select the Services tab.
- 2 Expands Server, right-click the GlassFish node, and then select Start.

### Viewing Service Engine Properties

After you have started the application server, you can access the service engine properties.

#### ▼ To View XSLT Service Engine Properties

- 1 In the Services tab, expand the following nodes: GlassFish V2 > JBI > Service Engines.
- 2 Right-click sun-xslt-engine and select Properties.  
The sun-xslt-engine Properties window appears.

## Runtime Property Descriptions

TABLE 1 General Properties

Property Name	Description	Default Value
Description	Description of the JBI Component.	XSLT Service Engine allows XSL stylesheets to be deployed as web services.
Name	Name of the JBI Component. Specifies a unique name in the JBI environment. If you are installing more than one XSLT Service Engine in a JBI environment, make sure that each is unique. This can be changed in the descriptor (jbi.xml) for the component. When the service unit deploys the component, it is matched with target component name defined in its descriptor — jbi.xml.	sun-xslt-engine
State	State of the JBI Component. Start, Stop, or Shutdown.	Started
Type	Type of the JBI Component (service-engine or binding-component)	service-engine

TABLE 2 Identification Properties

Property Name	Description	Default Value
Build Number	Date and time stamp for the current build.	<build_number>
Spec Version	XSLT specification fully supported by this build.	\$(specification version)

TABLE 3 Class Loggers

Class Name	Description	Default Logging Level
sun-xslt-engine		INFO
DefaultMessageListener		INFO
PatternExchangeRouter		INFO
SimpleProcessorFactory		INFO



TABLE 3 Class Loggers (Continued)

Class Name	Description	Default Logging Level
SingleThreadManager		INFO
xsltseComponentManager		INFO

## Configuring the XSLT Service Engine

No special knowledge of XSL is required to configure an XSLT service unit. The only requirements are a WSDL, an XSL style sheet, and the XSLT map configuration file.

The creation of the WSDL — defining the service, port type, and creating the binding information for the service definition — is a separate step from the XSLT map configuration file. Properly configuring an XSLT service unit means understanding what pieces to migrate from the WSDL into the XSLT map file.

XSLT Projects contain a configuration file called `xsltmap.xml`. Most of the information needed to configure an XSLT Project as a JBI service unit is contained in a WSDL, which defines the operation an XSLT transformation is implementing. A sample configuration follows:

### Use Case 1

```
<xsltmap>
  <requestReplyService>
    <input partnerLink="{foo}p10"
      roleName="server"
      portType="portType"
      operation="operation"
      messageType="{ns}msg-name"
      file="map.xml"
      transformJBI="false" />
  </requestReplyService>
  <!--Names partnerlink defined in deployed WSDL
    <-- Matches partnerLink role name
  <-- Matches partnerLink portType name
    <-- Operation this transformation implements
    <-- Identifies reply message definition in deployed WSDL
  <-- The transformation applied to request
  <-- See details below
```

### Use Case 2

```
<filterOneWay>
  <input partnerLink="{foo}p11"
    roleName="server"
    portType="portType"
    operation="operation"
    messageType="{ns}msg-name"
    file="map.xml" />
  <output partnerLink="{bar}p12"
    roleName="client"
    portType="outPortType"
    operation="outOp" />
</filterOneWay>
  <!--Names partnerlink defined in deployed WSDL
  <--Matches partnerLink role name
  <--Matches partnerLink portType name
  <--Operation this transformatio implements
  <--Identifies transformed message definition in
  deployed WSDL
  <--The transformation applied to request
  <--See details below
  <--Names partnerLink of operation to invoke
  <--Matches named partnerLink
  <--Matches portType of operation to invoke
  <--Operation to invoke/send transformed request
```

## Configuring the XSLT Service Engine

```

<filterRequestReply>
  <!--Names partnerlink defined in deployed WSDL
  <input partnerLink="{foo}p11"
    roleName="server"
    portType="portType"
    operation="operation"
    messageType="{ns}msg-name"
    file="map.xml" />
    transformJBI="false" />
  <output partnerLink="{bar}p12"
    roleName="client"
    portType="outPortType"
    operation="outOp" />
    messageType="{ns}msg-name"
    file="map2.xml"
    transformJBI="fales" />
</filterRequestReply>
</xsltmap>

```

### transformJBI attribute

To support multiple-part WSDL 1.1 message definitions, the JBI specification defines an XML document format for *wrapping* WSDL 1.1 message parts. Since the XSLT Service Engine does not lend itself to transforming multiple message parts, the XSLT Service Engine instead supports the transformation of the entire JBI message wrapper. To enable this transformation, the `transformJBI` attribute flag must be set to `true`; the default value is `false` if the attribute is not specified in the `xsltmap.xml` file.

It is important to note that when this attribute is set to `true`, the XSL style sheet **MUST** generate a properly formed JBI message wrapper document. Failure to do so will result in mishandled message exchanges. As noted in the Java Business Integration (JBI) 1.0 specification, the wrapping of message parts allows both consumer and provider to interact using this well-known mapping to a *wrapped doc-literal* form of the message that is used for normalized message content.

A wrapped literal document must conform to the schema given in the listing below.

```

default namespace jbi = "http://java.sun.com/xml/ns/jbi/wsdl-11-wrapper"
start =
element message {
  attribute version { xsd:decimal },
  attribute type { xsd:QName },
  attribute name { xsd:nmtoken }?,
  part*
}
part =
element part {

```

```
# part value  
( (element* { text } | text)  
}
```

See the Java Business Integration (JBI) 1.0 specification for additional information on the normalized message context schema for wrapper document for WSDL 1.1–defined messages.

## messageType attribute

To eliminate the need of parsing the WSDL to determine the output message definition (as it is required in a JBI message wrapper), this attribute must be specified for all non-JBI transformations. That is, if the `transformJBI` attribute is false, then the `messageType` attribute MUST be specified. The value of this attribute usually takes the form: `{msg-def-ns}msg-def`.

## XSLT Service Engine Component Limitations — Running the JBI Runtime Separately

Due to a dependency on the design-time XSLT project (part of the NetBeans IDE), XSLT service units used outside the NetBeans environment do not benefit from the functionality built into the XSLT project. Specifically, the generation of the service unit's `jbi.xml` file and the creation of a distributable/deployable service unit must be done manually.

