Oracle® Java CAPS XSLT Editor User's Guide

Part No: 821–2615
January 2011

ORACLE

120126@25097

Contents

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

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

Use Case 2

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

Configuring the XSLT Service Engine

```
<filterRequestReply>                <--Names partnerlink defined in deployed WSDL
              <input partnerLink="{foo}p11"            <--Matches partnerLink role name
                    roleName="server"                        <--Matches partnerLink portType name
                    portType="portType"                     <--Operation this transformatino implements
                    operation="operation"                  <--Identifies transformed message definition in
                                                    deployed WSDL
                    messageType="{ns}msg-name               <--The transformation applied to request
                    file="map.xsl" />                          <--See details below
                    transformJBI="false" />                <--Names partnerLink of operation to invoke
                <output partnerLink="{bar}p12"            <--Matches named partnerLink
                    roleName="client"                          <--Matches portType of operation to invoke
                    portType="outPortType"                    <--Operation to invoke/send transformed request
                    operation="outOp" />                   <--Identifies reply message definition in
                                                    deployed WSDL
                    messageType="{ns}msg-name"             <--Transformation applied to response from
                                                     invoked operation
                    file="map2.xsl"                            <--See details below
                    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.