JBoss ESB 4.2.1 GA

Message Transformation Guide

JBESB-MTG-10/31/07

Legal Notices

Copyright

Software Version

JBoss ESB 4.2.1 GA

Restricted Rights Legend

Contents

Table of Contents

About This Guide

What This Guide Contains

A common obstacle encountered during Enterprise Integration is that of bridging the gaps created by the fact that business constructs from different application domains representing the same data are typically represented through different data formats. Bridging this gap is one of the key features of JBoss ESB, as well as the subject of this guide.

Audience

This guide is most relevant to engineers who are responsible for using JBoss ESB 4.2.1 GA installations and want to know how it relates to SOA and ESB principles.

Prerequisites

None.

Organization

This guide contains the following chapters:

1. **Chapter 1, Overview:** An overview of the message transformation solutions provided by JBoss ESB.
2. **Chapter 2, Smooks:** Using the *Smooks Transformation Management Framework* to manage your message transformation logic.
3. **Chapter 3, XSL Transformations:** Performing message transformation XSLT.
4. **Chapter 4, Binary Format Transformations:** Performing binary format transformations.

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
Italic	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
`Code`	Text that represents programming code.
Function | Function	A path to a function or dialog box within an interface. For example, "Select File | Open." indicates that you should select the Open function from the File menu.
( ) and |	Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax: `persistPolicy (Never | OnTimer | OnUpdate | NoMoreOftenThan)`
Note: **Caution**:	A note highlights important supplemental information. A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following guides are available in the JBoss ESB 4.2.1 GA documentation set:

1. **JBoss ESB 4.2.1 GA** *Trailblazer Guide*: Provides guidance for using the trailblazer example.

2. **JBoss ESB 4.2.1 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.

3. **JBoss ESB 4.2.1 GA** *Programmers Guide*: How to use JBossESB.

4. **JBoss ESB 4.2.1 GA** *Release Notes*: Information on the differences between this release and previ ous releases.

5. **JBoss ESB 4.2.1 GA** *Administration Guide*: How to manage the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.2.1 GA should be directed to our support team.

Introduction

Overview

JBoss ESB supports message data transformation through a number of mechanisms:

1. **Smooks**: Milyn Smooks is a Fragment based Data Transformation and Analysis tool (XML and non-XML). It can also be thought of as a management tool that allows you manage transformations across your entire message set using techniques such as message profiling. It supports transformation logic implementation through raw Java, XSLT, FreeMarker, Groovy and more.

2. **XSLT**: JBoss ESB supports message transformation through the standard XSLT usage model.

3. **ActionProcessor Data Transformation**: Transformations involving binary data formats are most easily performed through implementation of the *org.jboss.soa.esb.actions.ActionProcessor* Java interface. The *org.jboss.soa.esb.actions* package (in the "Listeners" module) has a number of out-of-the-box *ActionProcessor* based transformers.

Smooks

Introduction

Message Transformation on JBossESB is supported by the SmooksTransformer component. This is an ESB Action component that allows the Smooks Data Transformation/Processing Framework to be plugged into an ESB Action Processing Pipeline.

A wide range of source (XML, CSV, EDI etc) andtarget (XML, Java, CSV, EDI etc) data formats are supported by the SmooksTransformer component. A wide range of Transformation Technologies are also supported, all within a single framework. See Smooks for more details.

Samples & Tutorials

1. A number of Transformation Quickstart samples accompany the JBossESB distribution. Check out the "transform_*" Quickstarts.
2. A number of tutorials are available online on the Smooks website. Any of these samples can be easily ported to JBossESB.

SmooksTransformer Configuration

The basic configuration of this action simply takes a "resource-config" property that references a Smooks configuration file. The resource-config property value is any valid URI based resource, as defined by the URIResourceLocator class.

```
<action name="transform"
            class="org.jboss.soa.esb.actions.converters.SmooksTransformer">
     <property name="resource-config" value="/smooks-config.xml" />
</action>
```

Input/Output Configuration

By default, this actions gets its input from (and sets it's output on) the "Default Message Body Location" (i.e. *Message.getBody().add(Body.DEFAULT_LOCATION...)*and *Message.getBody().get(Body.DEFAULT_LOCATION)*). These can be overridden by specifying "input-location" and/or "output-location" configuration properties on the action.

Java Output Configuration

As stated above, this action supports source (XML, CSV, EDI etc) to Java object transformation/binding. See the "Transform_XML2POJO*" quickstarts for examples of this and also check out the Smooks Tutorials.

The constructed Java object model(s) can be used as part of a model driven transform, or can simply be used by other ESB action instances that follow the SmooksTransformer in an action pipeline.

Such Java object graphs are available to subsequent pipeline action instances because they are attached to the ESB Message output by this action and input to the following action(s). They are bound to the Message instance Body (*Message.getBody().add(...)*) under a key based directly on the objects "beanId" (as defined in the Smooks Javabean config). This means that the objects are available through the ESB Message Body by performing *Body.get(beanId)* calls.

The full Java object Map can also be made available on the output message by adding a "java-output-location" property e.g.

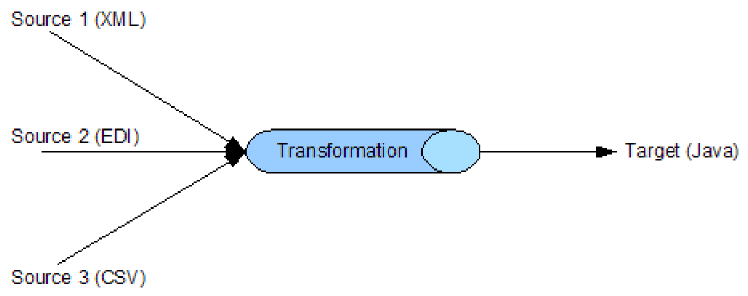
```
<action name="transform"
           class="org.jboss.soa.esb.actions.converters.SmooksTransf
ormer">
  <property name="resource-config" value="/smooks-config.xml" />
  <property name="java-output-location" value="order-message-
objects-map"/>
</action>
```

Or, shorthand for binding the map to the "Default Message Body Location":

```
<action name="transform"
           class="org.jboss.soa.esb.actions.converters.SmooksTransf
ormer">
    <property name="resource-config" value="/smooks-config.xml" />
    <property name="java-output-location" value="$default" />
</action>
```

Profile Based Transformation

In the following example we have messages being exchanged between 3 different Sources and 1 Target. The ESB needs to transform the messages supplied by each of the 3 Sources (supplied in different formats) into Java Objects before supplying them to the "Target" Service. Basically,we've 3 possible transformations for messages being exchanged to the Target. We'll call these transformations "source1", "source2" and "source3" respectfully.



From an ESB configuration perspective, we have a single Service configuration for the "Target" Service. The configuration might look something like the following.

```
<service category="ServiceCat" name="TargetService"
                                     description="Target Service">
    <listeners>
        <jms-listener name="Gateway-Listener"
                        busidref="quickstartGwChannel" is-
gateway="true"/>
        <jms-listener name="Service-Listener"
                        busidref="quickstartEsbChannel"/>
    </listeners>
    <actions>
        <!--
            A SmooksTransform action to transform the source
message into
            the Target Java Object(s)
         -->
        <action name="transform"
            class="org.jboss.soa.esb.actions.converters.SmooksTrans
former">
            <property name="resource-config" value="/smooks-
config.xml"/>
        </action>

        <!-- An action to process the Java Object(s) -->
        <action name="process" class="com.acme.JavaProcessor" />

    </actions>
</service>
```

As can be seen by the SmooksTransformer configuration, we only define a single transformation configuration file called "/smooks-config.xml". We need to define 3 different transformation, one for each source. This is done using a Smooks Message Profiling.

Basically, we define the 3 separate transformations in 3 separate Smooks configuration files (from_source1.xml, from_source2.xml and from_source3.xml) and <import> them into the top-level smooks-config.xml. In each of these configuration files we specify the "default-target-profile" for that configuration set e.g.

```
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.0.xsd"
                  default-target-profile="from:source1">
    <!--
        Source1 to Target Java message transformation
        resource configurations...
    -->
</smooks-resource-list>
```

The top-level smooks-config.xml looks as follows:

```
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.0.xsd">
    <import file="classpath:/from_source1.xml" />
    <import file="classpath:/from_source2.xml" />
    <import file="classpath:/from_source3.xml" />
</smooks-resource-list>
```

So what this does (effectively) is to load a single SmooksTransformer instance with 3 different transformations, each transformation defined under it's own unique "profile" name. There are actually more uses for message profiling than this, but this view of profiling works fine for this example.

In order for the SmooksTransfromer to know which of the 3 transformations needs to be applied on a given message, the message (ESB Message) needs to have it's "from" property set before the message flows into the SmooksTransformer. This can be done anywhere that makes sense - at the Source itself (source1, source2 etc), or in a content based action that precedes the SmooksTransformer. See details of the transform_XML2POJO2 quickstart (below).

JBossESB also supports profiles additional to the "from" profile, namely "from-type", "to" and "to-type". These can be used in combination, leading to more intricate exchange based transforms - n possible inputs to n possible outputs, sharing profile sets etc. See the "profiling" tutorial on the Smooks website.

Transform_XML2POJO2

The basic scenario outlined above is implemented as a quickstart within the JBossESB distribution. The quickstart is named "transform_XML2POJO2". In this quickstart there are 2 message sources. It utilises a Groovy script on the action pipeline to detect and set the "from" profile for the incoming message.

- jboss-esb.xml: JBossESB Configuration File.
- smooks-config.xml: Top Level Transformation Configuration.
- from-dvdstore.xml: DVD Store message Transformations Configuration - imported into top-level smooks-config.xml (notice the profile configuration?). Designed to transform a DVD Store message into Java Objects.
- from-petstore.xml: Pet Store message Transformations Configuration - imported into top-level smooks-config.xml (notice the profile configuration?). Designed to transform a Pet Store message into Java Objects (the same Java object model).
- check-origin.groovy: Simple Groovy script for checking the origin of the message based on it's content.

XSL Transformations

following sections illustrate how to create Smooks Resource

Introduction

In this release of JBossESB, XSL Transformations are supported through Smooks. In later releases we will be supporting XSLT natively. Support for XSLT can be provided by creating a custom *org.jboss.soa.esb.actions.ActionProcessor* implementation.

Binary Format Transformations

Introduction

Binary Transformations are currently supported through custom implementation of the *org.jboss.soa.esb.actions.ActionProcessor* interface.

JBossESB is shipped with a number of out-of-the-box binary transformers e.g. *org.jboss.soa.esb.actions.ObjectToXStream* and *org.jboss.soa.esb.actions.ObjectToCSVString*.