JBoss ESB 4.2 GA

Message Transformation Guide

JBESB-MTG-8/29/07

Legal Notices

The information contained in this documentation is subject to change without notice.

JBoss Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. JBoss Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Java™ and J2EE is a U.S. trademark of Sun Microsystems, Inc. Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation. Oracle® is a registered U.S. trademark and Oracle9™, Oracle9 Server™ Oracle9 Enterprise Edition™ are trademarks of Oracle Corporation. Unix is used here as a generic term covering all versions of the UNIX® operating system. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright

JBoss, Home of Professional Open Source Copyright 2006, JBoss Inc., and individual contributors as indicated by the @authors tag. All rights reserved.

See the copyright.txt in the distribution for a full listing of individual contributors. This copyrighted material is made available to anyone wishing to use, modify, copy, or redistribute it subject to the terms and conditions of the GNU General Public License, v. 2.0. This program is distributed in the hope that it will be useful, but WITHOUT A WARRANTY;without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details. You should have received a copy of the GNU General Public License, v. 2.0 along with this distribution; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Software Version

JBoss ESB 4.2 GA

Restricted Rights Legend

Use, duplication, or disclosure is subject to restrictions as set forth in contract subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause 52.227-FAR14.

© Copyright 2007 JBoss Inc.

Contents

Table of Contents

Contents..... iv

About This Guide..... 5

What This Guide Contains..... 5

Audience..... 5

Prerequisites..... 5

Organization..... 5

Documentation Conventions.....6

Additional Documentation.....7

Contacting Us..... 7

Introduction..... 8

Overview..... 8

Smooks..... 9

Introduction..... 9

The Smooks Transformation Model.....9
Message Profiles..... ..10
Integration..... 11
SmooksTransformer Configuration.....11
Smooks Resource Configurations.....12
Manual Resource Configuration.....13
JBoss ESB Administration Console.....13
Console Home Page.....14
Use Case 1: Defining an XSLT Fragment Transformation.....16

Introduction..... 29

Binary Format Transformations..... .30

Introduction..... 30

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 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 GA documentation set:

1. **JBoss ESB 4.2 GA** *Trailblazer Guide*: Provides guidance for using the trailblazer example.
2. **JBoss ESB 4.2 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.
3. **JBoss ESB 4.2 GA** *Programmers Guide*: How to use JBossESB.
4. **JBoss ESB 4.2 GA** *Release Notes*: Information on the differences between this release and previous releases.
5. **JBoss ESB 4.2 GA** *Administration Guide*: How to manage the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.2 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, 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

Milyn Smooks is integrated into JBoss ESB as one of it's out-of-the-box data Transformation solutions. Some of the terms and concepts presented in this section of the document may require you to refer to the Smooks documentation.

As outlined on the Milyn Smooks website, Smooks is a framework for both "Implementing" and "Managing" a flexible XML (and non-XML) data Transformation solution across an enterprise message set.

From an "implementation" perspective, Smooks' goal is to provide a framework in which messages can be transformed using the transformation technology most appropriate to the transformation required e.g. it may be easier to transform one part of the message using XSLT and another part using raw Java code.

From a "management" perspective, Smooks' goal is to provide a framework in which an "enterprise message set" can be "profiled", with transformations being applied based on these profiles. In this way, transformation resources can be more easily reused and transformation configuration can be reduced. More on message profiles later.

Smooks achieves its goals by applying a processing model different from that applied by the likes of XSLT.

The Smooks Transformation Model

With XSLT, message transformation is performed in batch and "targeted" at a "whole" document i.e. at the document level. In general, XSLT transformations are thought about, defined and applied with the whole message in mind.

Smooks applies (targets) transformation logic at the element level i.e. at the document fragment level. Smooks transforms the "whole" message through the application of multiple "mini" transformations that have been "targeted" at specific message fragments. Smooks uses a message's "profile set" to decide whether or not an individual "mini" transformation should be applied to a given message fragment. So what is a message "profile"?

Message Profiles

Central to how Smooks functions is the concept of a message "profile". A message profile is simply a unique name applied to a group of messages to which common transformation resources[1] need to be targeted.

So how does the concept of a message profile map into the world of JBoss ESB Transformations? Within the context of JBoss ESB, message transformations are performed in order to make a messages "produced" by one entity "*A*", "consumable" by another entity "*B*". We refer to this interaction as a "Message Exchange", with entities "*A*" and "*B*" being the "Message Exchange Participants". Therefore, all JBoss ESB Transformations are defined in terms of a "Message Exchange". They are defined with respect to the two Participants involved in that Message Exchange, as well as the message typing information associated with the message being exchanged between the 2 participants[2]. Therefore, the 4 properties of all Message Exchanges are:

1. **from**: Message producer participant name,
2. **from-type**: Message producer message type (message type produced),
3. **to**: Message consumer participant name,
4. **to-type**: Message consumer message type (message type consumed),

It is these 4 properties that map JBoss ESB messages into the Smooks world of "Message Profiles". Each one of these properties translates directly into a message profile.

As you can imagine, multiple messages can potentially share an intersecting subset of profiles. Using the 4 message exchange based message profiles above , JBoss ESB can specify and target all transformations that need to be applied to all messages "from" (produced by) a particular message exchange participant using a single set configurations. In this case, the details relating to the transformation requirements of the message consumer (where the message is going to "to") can be handled by a separate set of configurations targeted at the "to" and/or "to-type" message exchange profiles.

[1] The definition of a "transformation resource" applies to anything that can be targeted at a message profile and used in the course of that message's transformation. Typically this will be units of transformation "logic" (XSLT, Java), but is not limited to that. They can also be things like profile-based parameters or configurations.

[2] Note that any message exchange participant can potentially produce or consume multiple message types

Integration

Smooks is integrated into JBoss ESB through an ActionProcessor class called *org.jboss.soa.esb.actions..convertersSmooksTransformer.*

SmooksTransformer Configuration

SmooksTransformer actions are configured into a listener action pipeline in the standard way as follows:

```
<action class="org.jboss.soa.esb.actions.converters.SmooksTransformer"
        from-type="Acme-Order-XML"
        from="Acme"
        to-type="PartnerX-Order-XML"
        to="PartnerX"
        />
```

From this you can see that the *SmooksTransformer* requires 4 properties, corresponding directly with the JBoss ESB message exchange profiles mentioned earlier. How to create these Transformation Resource Configurations is outlined in the following sections.

Smooks Resource Configurations

Smooks is configured (transformation resources are targets) using a set of XML configuration files listed in a file called "*smooks-cdr.lst*". The folder containing this file must be specified in the classpath i.e. this file must be in the root of the classpath.

The *smooks-cdr.lst* file simply specifies a set of Smooks resource configuration URIs to be loaded into the Smooks context. Each list entry URI can load a configuration set over the standard protocols supported by the *java.net.URL* class ("http", "file" etc), with the addition of support for loading resources from the classpath. The following is a simple example of the *smooks-cdr.lst* file:

```
# Load from the classpath – note "classpath" is the default protocol...
/com/acme/integr/smooks/messages_from_partnerx.cdrl
classpath:/com/acme/integr/smooks/messages_to___partnerx.cdrl

# Load over http – from the JBoss ESB Admin Console...
http://localhost:8080/jboss-esb-console/transform/smooks-config.jsp
```

The individual files should not be interpreted as being separate transformations; this is purely a configuration grouping scheme. Having them all in a single XML file might not be so easy to navigate or maintain. It is the combined contents of these files that matters. They constitute an effective Smooks Configuration "Database".

JBoss ESB offers 2 methods of creating Smooks Configurations:

1. Manual creation of the XML configurations and their listings in the *smooks-cdr.lst* file.
2. Through the **JBoss ESB Administration Console**.

Both these options are available to JBoss ESB at the same time i.e. you can have most of your configurations maintained through the **JBoss ESB Administration Console**, while at the same time having a small subset maintained manually. This is possible simply because Smooks (and therefore the *SmooksTransformer*) loads it's configurations as XML streams via the URI set listed in the *smooks-cdr.lst* file, with the **JBoss ESB Administration Console** making the configuration set under its control available over HTTP (`http://<host>:<port>/jboss-esb-console/transform/smooks-config.jsp`). As outlined above, HTTP is one of the protocols supported by the *smooks-cdr.lst* file.

Manual Resource Configuration

The best reference on manual creation of Smooks Resource Configurations can be found on the Milyn Smooks website.

JBoss ESB Administration Console

The **JBoss ESB Administration Console** is the easiest way to create Smooks Resource Configurations. This application is a J2EE Web Application. This application persists all configurations to a database using the Hibernate persistence framework. The application currently **runs on JBoss Application Server ONLY**[3].

In order to use the ESB Administration Console, it needs to be packaged (from the release distribution) for your local environment. Here are the steps to packaging, deploying and configuring the console application[4]:

1. Open a command terminal.
2. Change directory to the "tools" folder of your expanded JBoss ESB distribution.
3. run "**ant**"[5].
4. Answer the environment specific questions presented by the script.
5. On completion of the script execution, copy "**jboss-esb-console.war"** from the root of the distribution "tools" folder to your JBoss App Server "deploy" folder. If this script produces a "**jboss-esb-console-ds.xml**" Datasource descriptor file, be sure to also copy this file to the JBoss App Server "deploy" folder.
6. Once the application is deployed and running, the required database tables will be automatically created in the target Database (by Hibernate). Populate these database tables by running the tools/*CONSOLE-import.sql* script. If deploying to Hypersonic, this step is not required.
7. That's it!

Direct your browser to the following URL:

http://<host>:<port>/jboss-esb-console/

There are a number of useful Flash demonstrations available online at JBoss Labs. Some of these Flash demos are also packaged inside the Administration Console application itself – available on the "About" page.

[3]Tested against version 4.0.5.GA of the JBoss Application Server (EJB3 installed).
[4]Note that these instructions are specific to building the console deployment from a release distribution i.e. not from source.
[5]Before running the "ant" command to package the console application for your environment, be sure to set the JAVA_HOME(JDK5) and ANT_HOME (v1.6.5) settings in your environment.

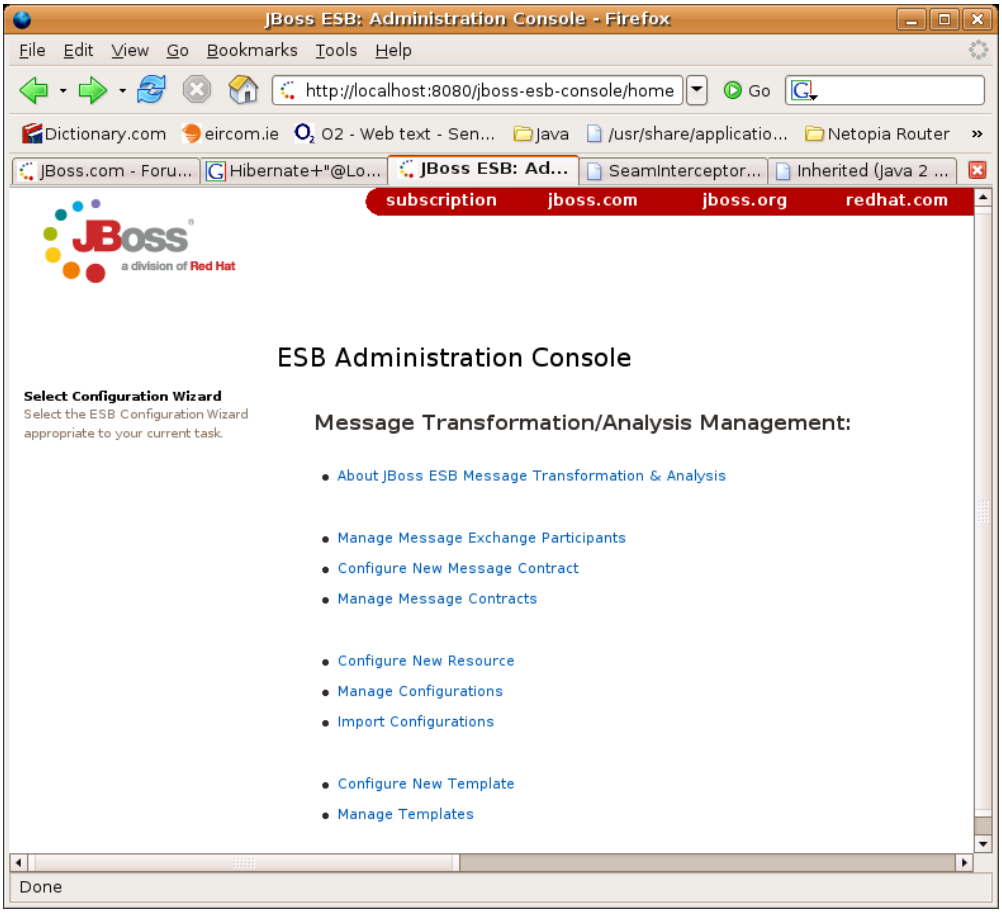
Smooks Configurations maintained by the Administration Console can be loaded by the *SmooksTransformer* by specifying the following URL in the *smooks-cdr.lst* file:

http://<host>:<port>/jboss-esb-console/transform/smooks-config.jsp

The following sections illustrate how to create Smooks Resource Configurations through the Administration Console.

Console Home Page

The Console Home appears as follows:



The basic steps involved in **defining** and **targeting** any Smooks Configuration through the Admin Console are the same:

1. **Create the Message Exchange Participants**. This is done through the "Manage Message Exchange Participants" wizard off the console main menu. Obviously, this only needs to be done once for the lifetime of each participant. If the participant is a Service, use the Service Logical Name. Otherwise, simply make up a meaningful unique name for the Participant (e.g. the component name).

2. **Configure the Message Contracts** associated with each Message Exchange Participant . This is done through the "Configure New Message Contract" wizard off the console main menu. This is basically a process of defining the types of messages that a Participant can produce or consume[6].
3. **Set the active Message Exchange** for the browser session. This sets the Message Exchange to be associated with all future transformation configurations made on this browser session. If not already set, the console will automatically redirect the user to set the active Message Exchange.
4. **Configure and target the new Resource**. This is a 4 step process:
   - Select "Configure New Resource" from the console main menu.
   - Select a template upon which to base the new configuration. For example, if you simply wish to specify an XSLT transformation, select one of the XSLT Resource templates from the list.
   - Fill in the template. This may include setting execution parameters for the resource.
   - Target the new resource configuration at a Message Exchange (and Fragment within the message). The Message Exchange details will be pre-populated based on the active Message Exchange selection, but can be manually tweaked in order to widen or narrow the selection.

Obviously, steps 1 and 2 do not have to be repeated for every configuration, as well as the fact that the Message Exchange is set on the browser session. Therefore, the typical usage scenario, for the most part, would involve multiple iterations of step 4.

[6] Note that a Participant can potentially produce or consume more than 1 message type.

Use Case 1: Defining an XSLT Fragment Transformation

This usage example will outline how to define an XSL Transform for an XML message fragment and target the transform at the message fragment for a specific "Message Exchange".

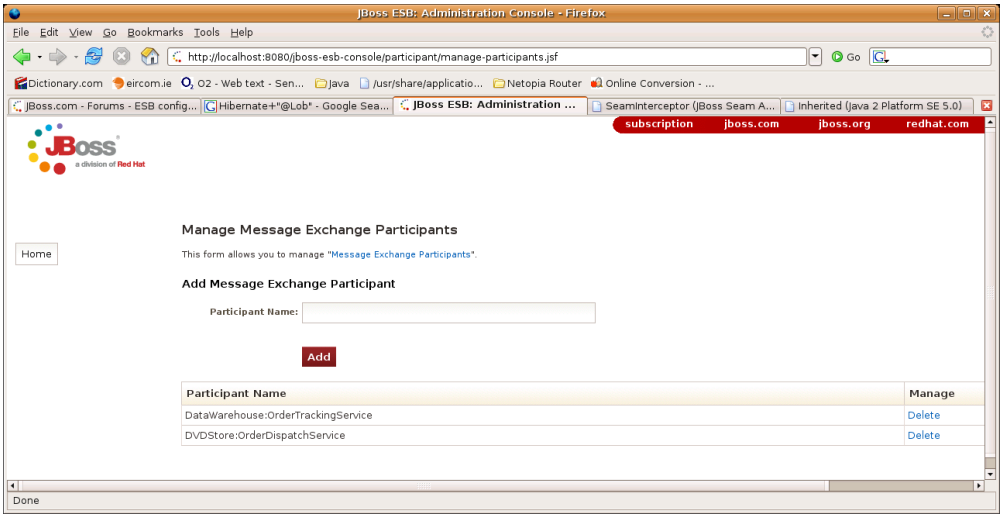
So in this example, the message exchange properties are as follow:

```
from-type="text/csv:fullFillOrder"
from="DVDStore:OrderDispatchService"
to-type="text/xml:recordOrder"
to="DataWarehouse:OrderTrackingService"
```

We have a Coma Separated Value ("text/csv:fullFillOrder") message produced by "DVDStore:OrderDispatchService" which needs to be transformed into a "text/xml:recordOrder" message for consumption by "DataWarehouse:OrderTrackingService".

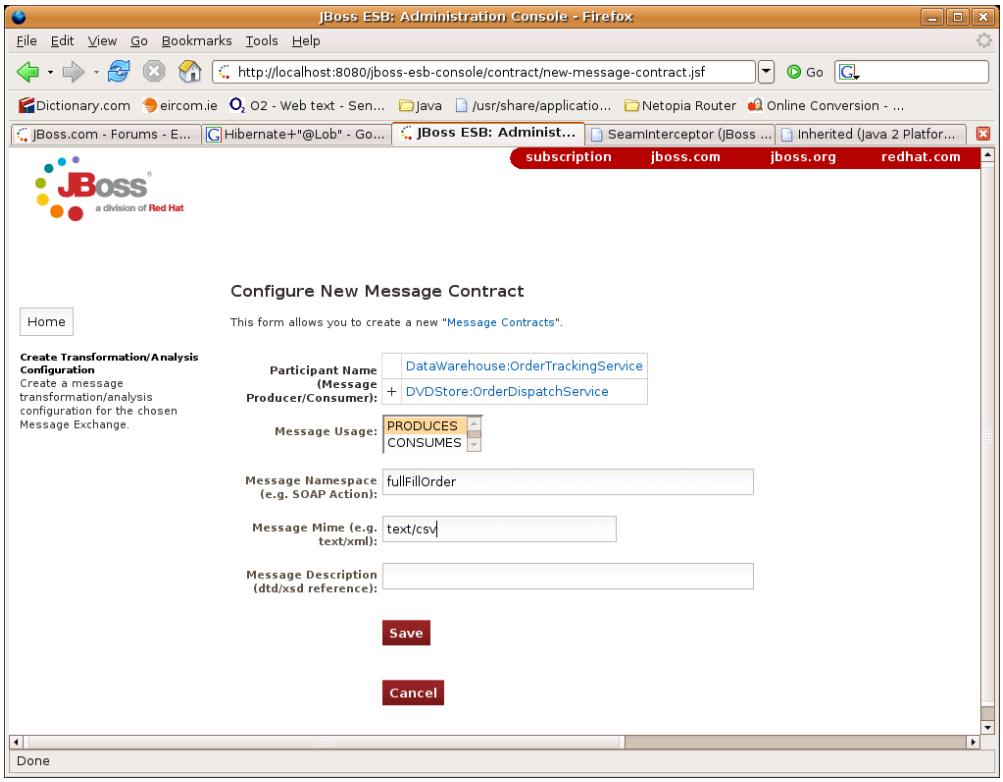
This means we need to define 2 Message Exchange Participants ("DVDStore:OrderDispatchService" and "DataWarehouse:OrderTrackingService") and 2 Message Contracts associated with each of these Participants.

The Message Exchange Participants are created and maintained through the "Manage Message Exchange Participants" wizard:

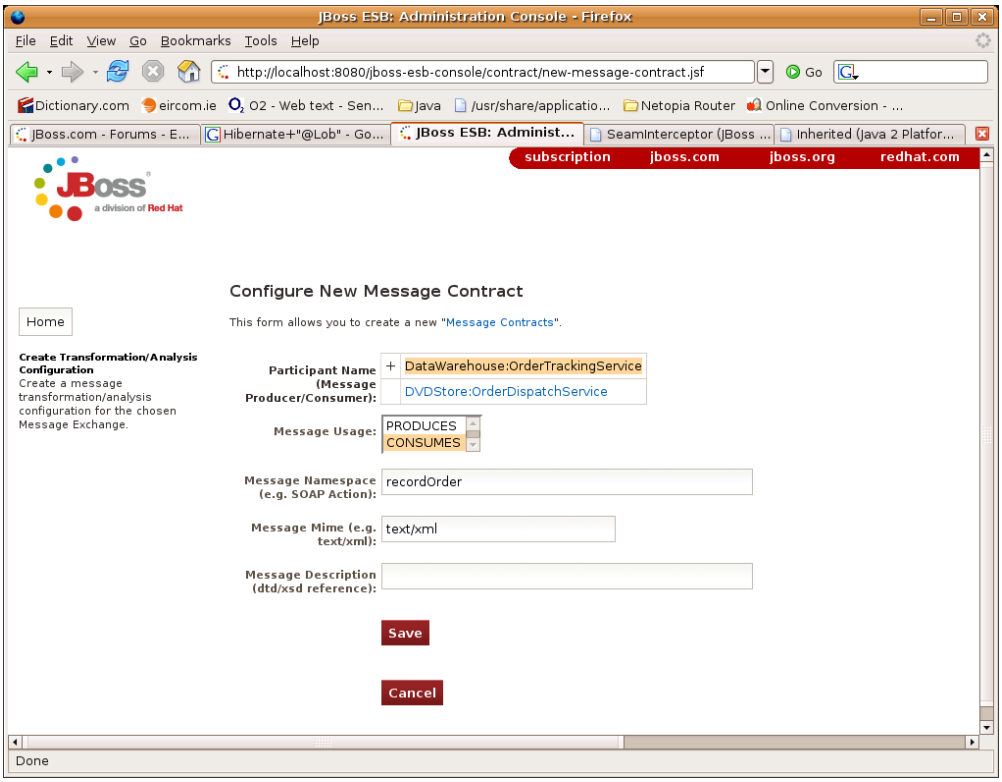


Once the Message Exchange Participants are defined, the Message Contracts associated with these Participants can be defined through the "Configure New Message Contract" wizard.

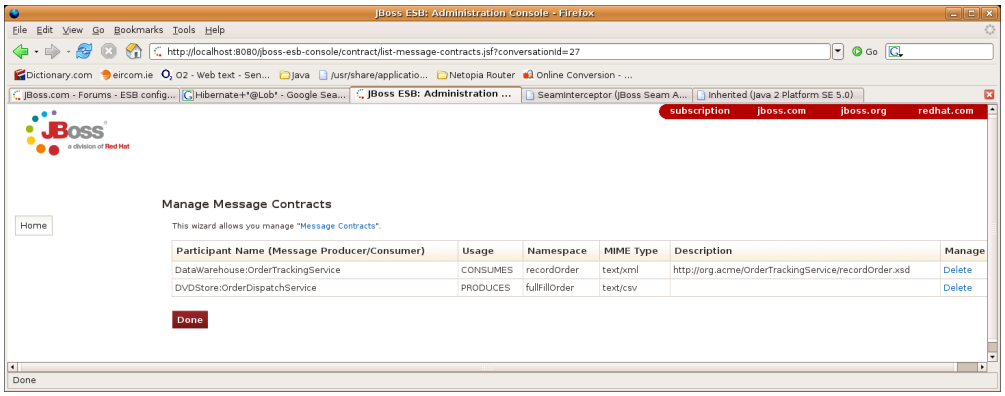
Configure the contract for the “text/csv:fullFillOrder“ message “PRODUCED” by “DVDStore:OrderDispatchService”:



Configure the contract for the “text/xml:recordOrder“ message “CONSUMED” by “DataWarehouse:OrderTrackingService”:

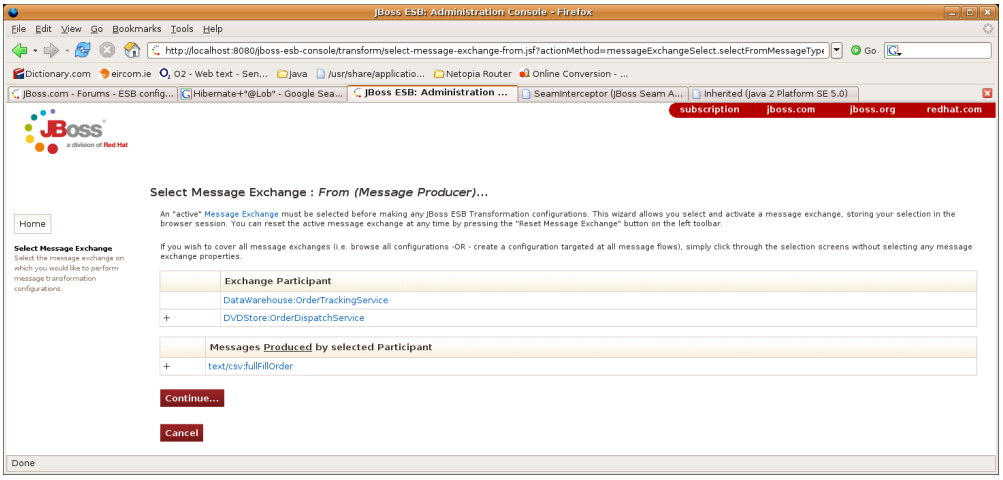


The new Message Contracts can then be reviewed from the “Manage Message Contracts” view:

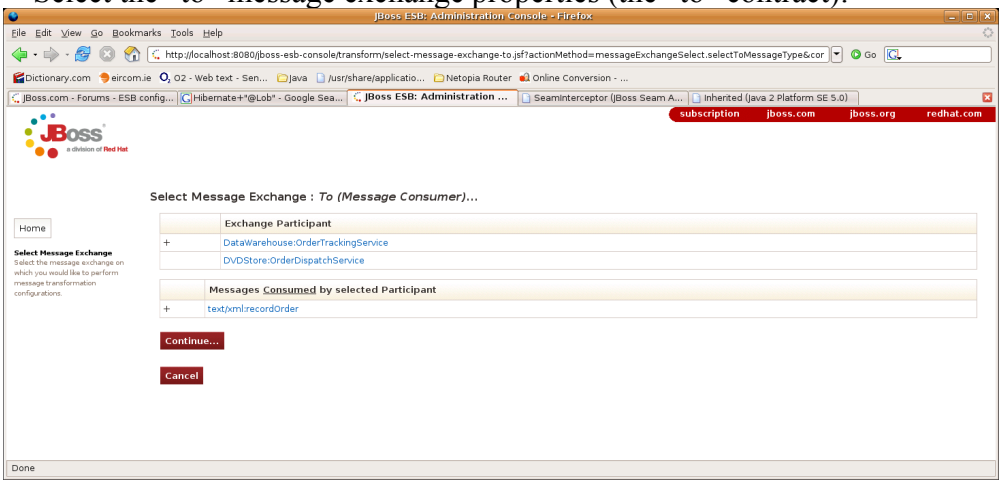


We are now ready to add the XSLT Transform and target it at the message fragment on the Message Exchange. We do this through the “Configure New Resource” wizard. At this point however, we haven't set the active Message Exchange for the browser session. To do this simply click on the link to access the “Configure New Resource” wizard. You'll notice that instead of this wizard being shown, you will be redirected to set the Message Exchange for the browser session. Just click your way through the screens to select the message exchange we wish to target (click the links and then press the “Continue..” button):

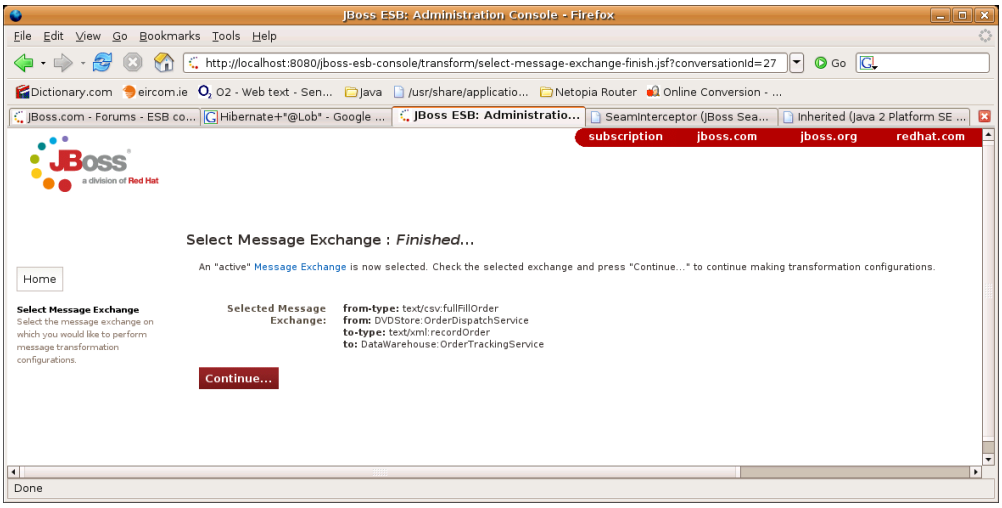
Select the “from” message exchange properties (the from contract):



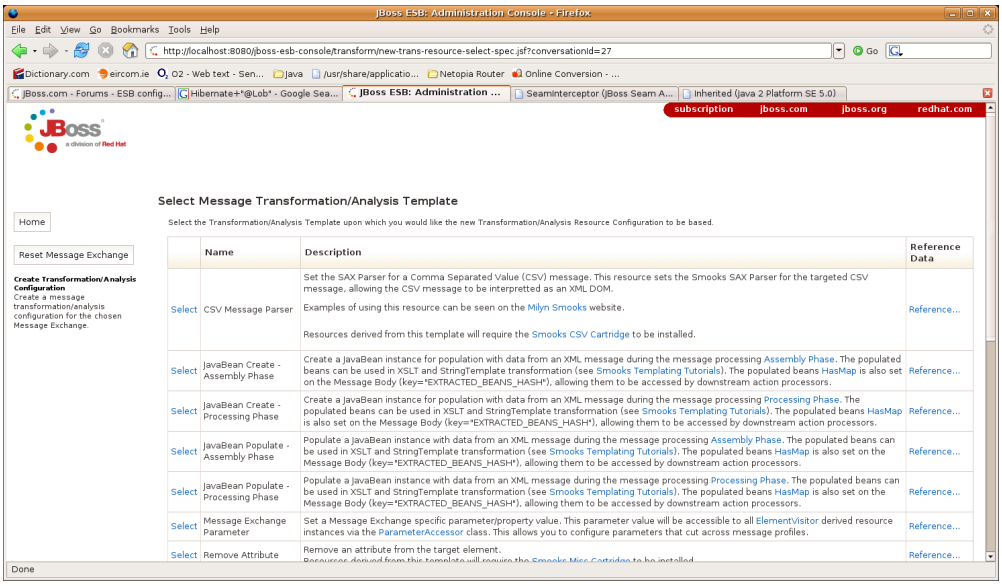
Select the “to” message exchange properties (the “to” contract):



Verify the selected message exchange and press “Continue” to start creating the resource configurations:



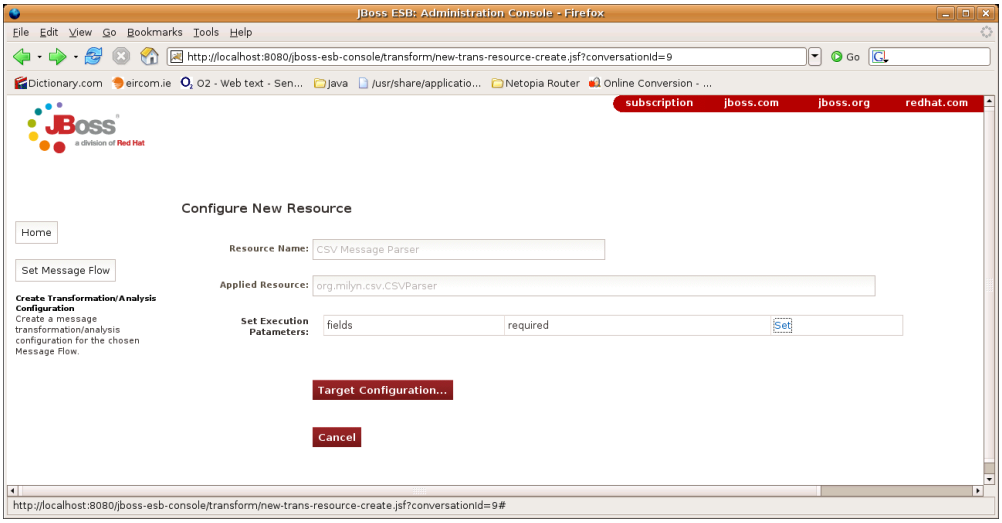
After verifying the active message exchange and pressing “Continue”, you'll be brought to the “Select Message Transformation/Analysis Template” page.



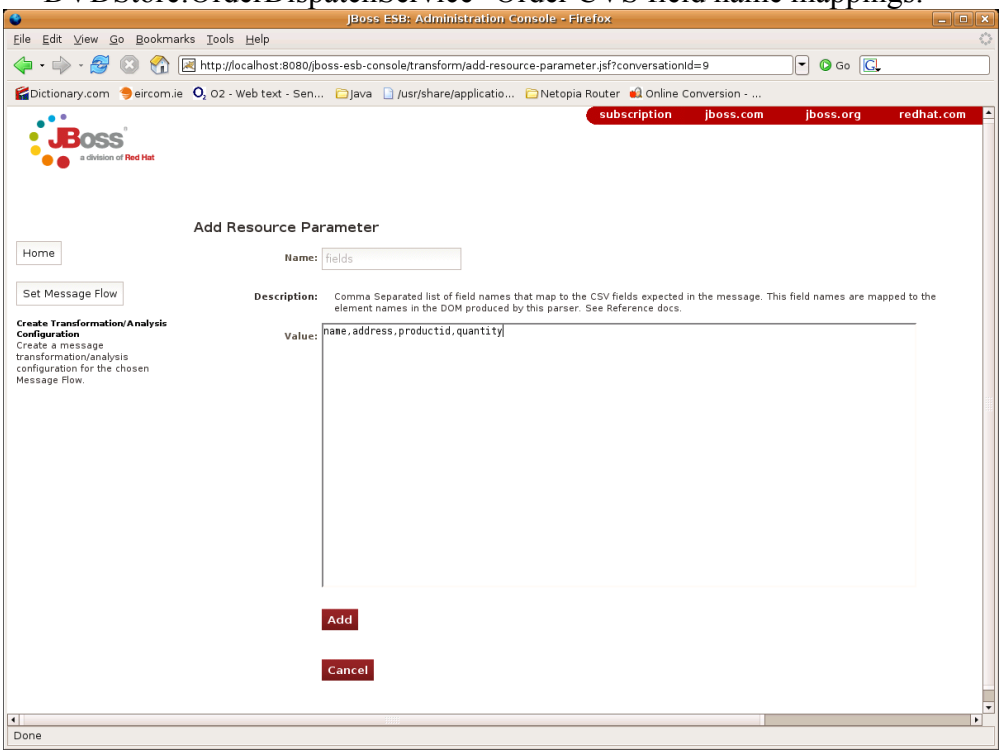
As outlined above, the first step in creating a transformation configuration is always that of selecting a configuration template upon which to base the configuration. However, before continuing, we need to decide on how to handle this message to get it from the CSV format produced by “DVDStore:OrderDispatchService”, to the XML format consumed by the “DataWarehouse:OrderTrackingService”. To do this we will need to configure 2 resources and target them at the Message Exchange:

1. **A CSV SAX Parser Configuration**: This will “suck” the CSV record(s) into a W3C DOM, allowing the records to be transformed into the “DataWarehouse:OrderTrackingService” XML format using XSLT.
2. **An XSLT Configuration**: This will perform the transformation on the DOM produced by the CSV Parser to produce the “DataWarehouse:OrderTrackingService” XML. In this case, the “fragment” we will target will be the whole message, simply by targeting the root of the DOM.

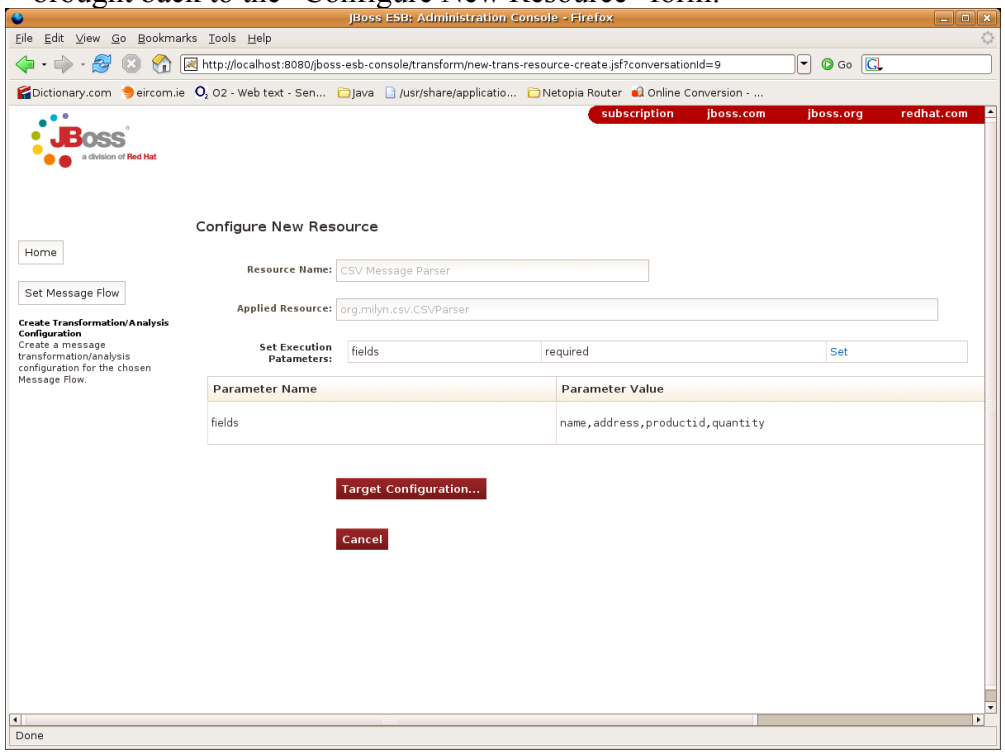
To configure a CSV Parser for this message, select the “CSV Message Parser” template (1st in list). This will bring you to the “Configure New Resource” form, from where you can configure the CSV fields, and target the Parser Configuration at the Message Exchange as follows:



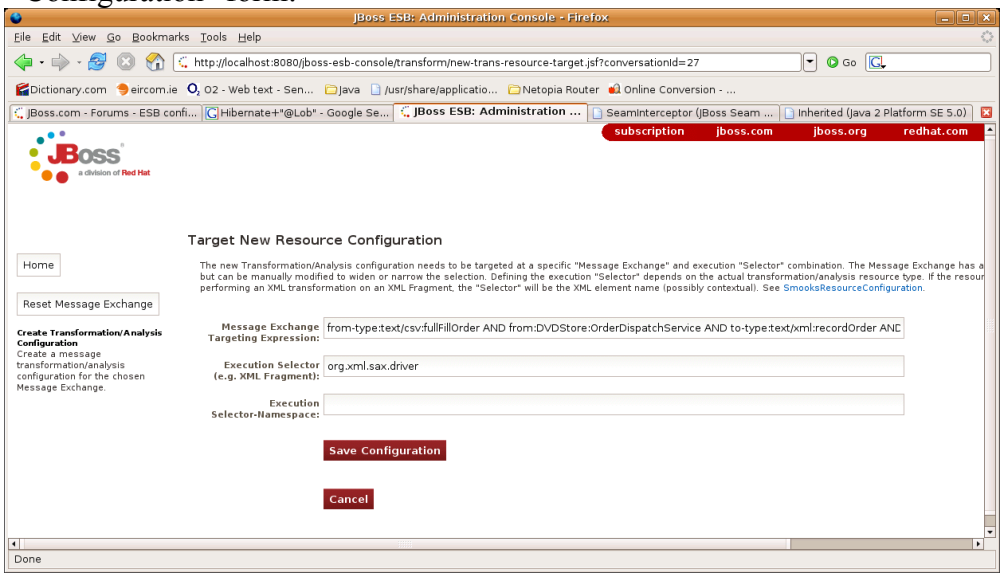
Click “Set” on the “fields” execution parameter and enter the “DVDStore:OrderDispatchService” Order CVS field name mappings:



Once the “fields” parameter has been added to the configuration you will be brought back to the “Configure New Resource” form:

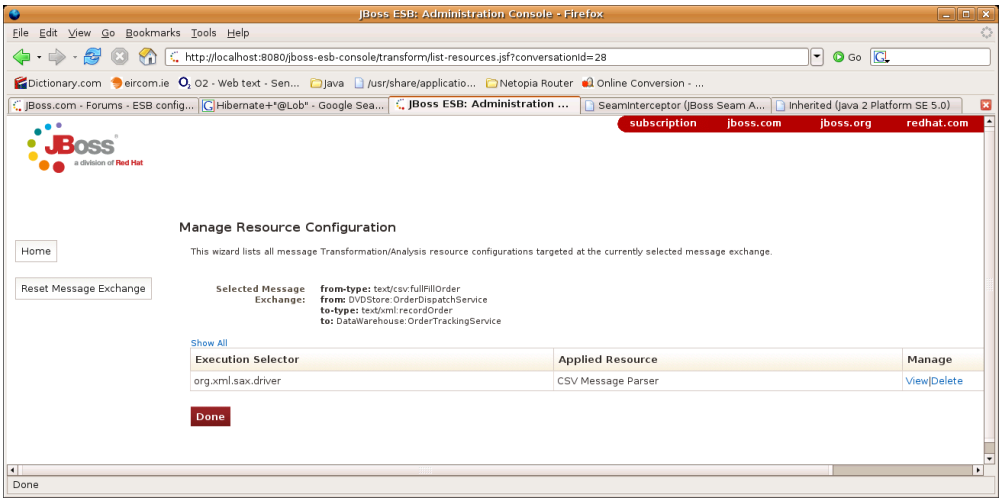


Press “Target Configuration...” to move on to the “Target New Configuration” form:



Note how the "Message Exchange Useragent Expression" is pre-populated with the details of the selected message exchange[7]. Just press "Save Configuration" to complete the process of creating the CSV Parser Smooks Configuration. You will return to the Console Home Page.

Select "Manage Configurations" and you will be able to see the new configuration:

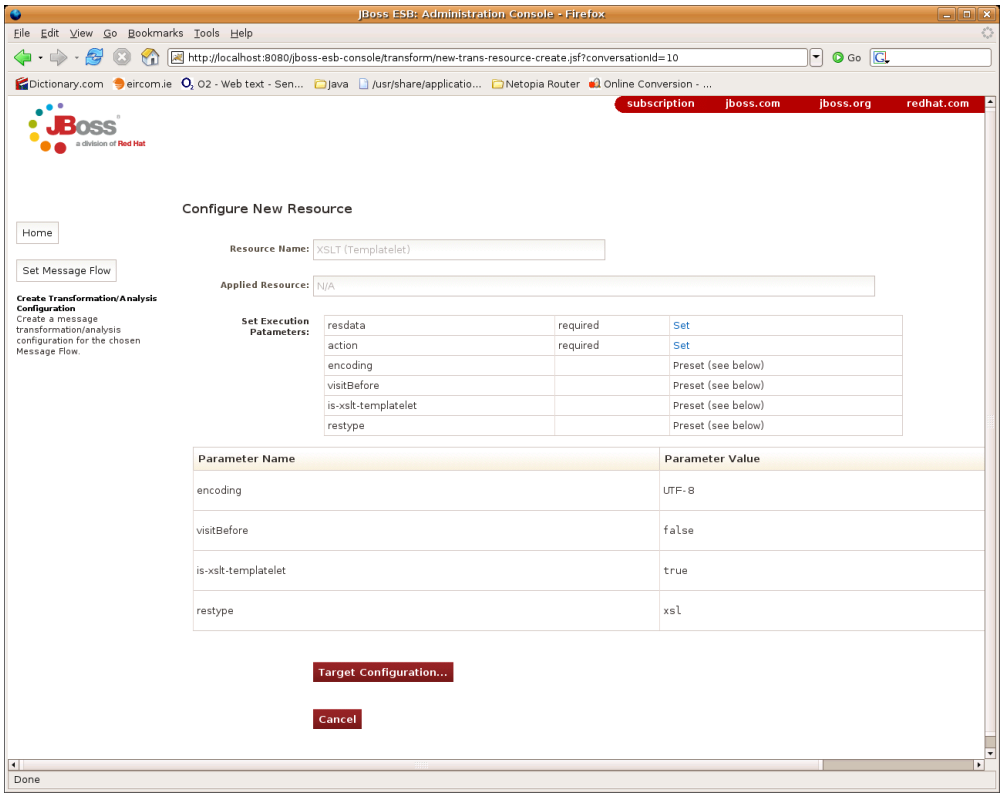


Next we need to create the XSLT configuration and target it at the same Message Exchange. Note that when this resource is applied by Smooks, the CSV message will be in a W3C DOM. The XML representation of this DOM will be of the following form (see Smooks CSVParser Docs):

```
<cvs-set>
        <cvs-record>
                <name>Tom Fennelly</name>
                <address>Ireland</address>
                <productid>V1234</productid>
                <quantity>3</quantity>
        <cvs-record>
        <cvs-record>
                <name>Joe Bloggs</name>
                <address>England</address>
                <productid>D9123</productid>
                <quantity>7</quantity>
        <cvs-record>
</cvs-set>
```

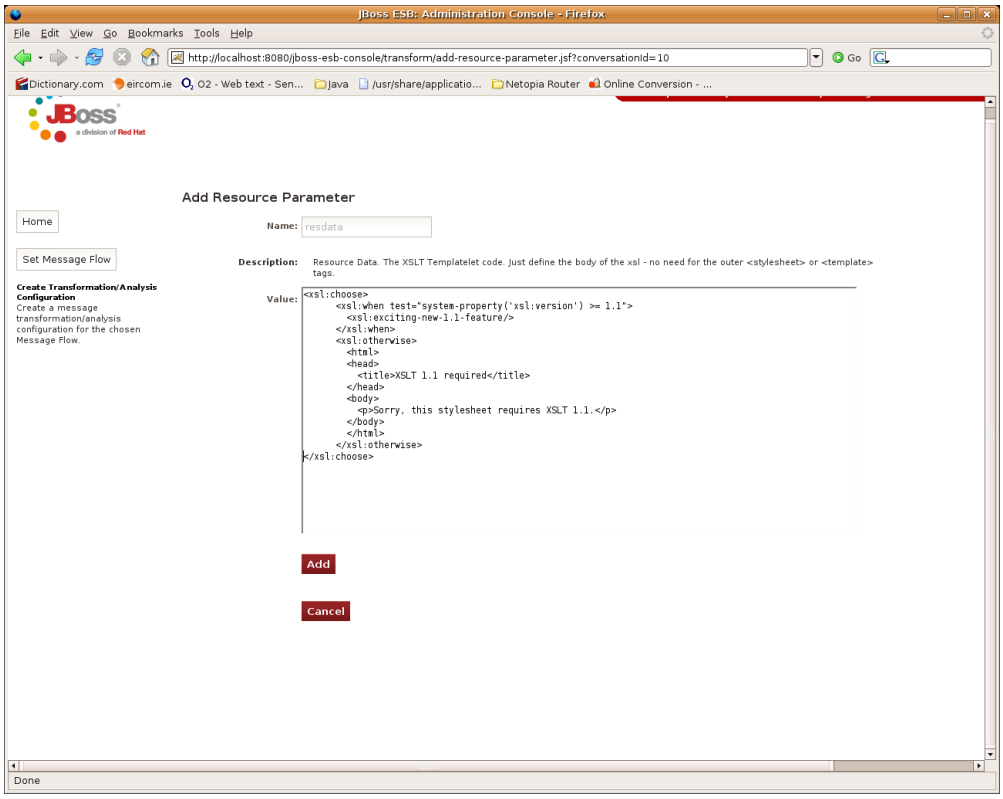
[7]Note that we could target the CSV Parser configuration at just the 1st half of the message exchange i.e. to cover all message exchanges where the source message is a "text/csv:fullFillOrder" message produced by "DVDStore:OrderDispatchService".

To configure an XSLT Transform Configuration to transform this message on the selected message exchange, click to access the “Configure New Resource” wizard again (screen shot above). This time select the “XSLT (Templatelet)” template (towards the end of the list). Once you select the template you'll be brought to the “Configure New Resource” wizard, from where you can fil in the template:



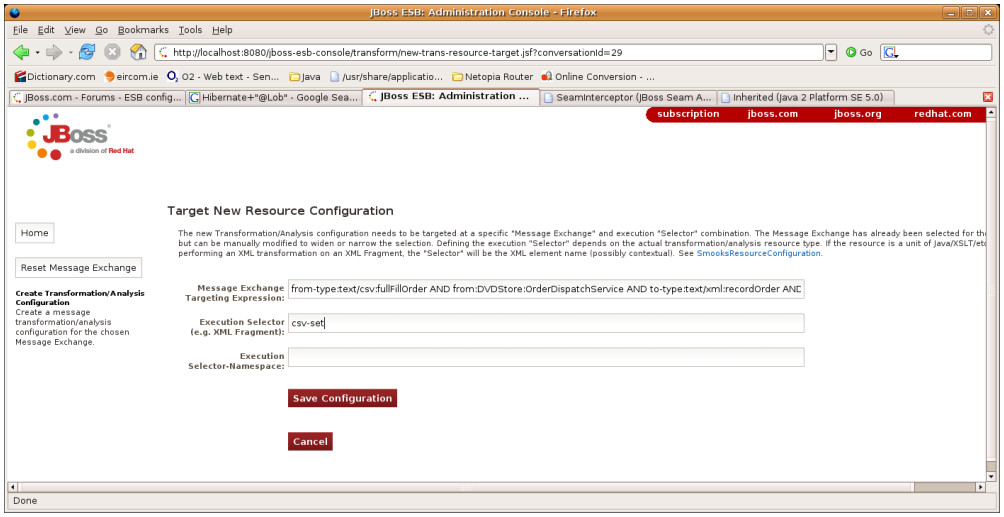
On this resource you need to set 2 Execution Parameters: “resdata” and “action”.

Set the “resdata” parameter. Note, with the XSLT Templatelet, there's no need to define all the stylesheet – see Smooks docs:



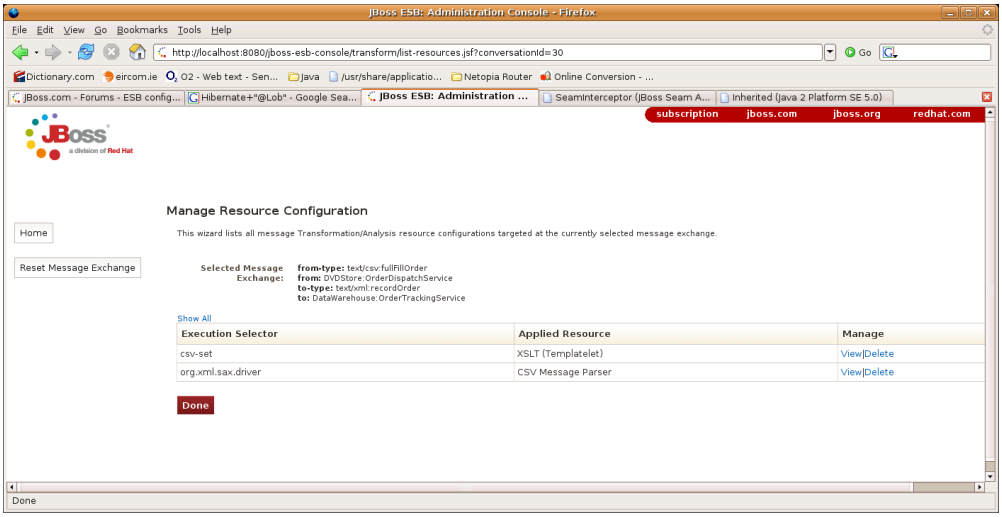
Set the “action” parameter to “replace” (see Smooks docs).

After adding both Execution Parameters, press the “Target Configuration...” button and move on to the “Target New Resource Configuration” form. This time you'll notice that the “Execution Selector” field is empty and needs to be populated. We want to target this XSL Transformation resource at the root the CSV Order DOM i.e. the “csv-set” element:



Note again that the “Message Exchange Useragent Expression” is pre-populated for the current Message Exchange.

Press “Save Configuration” to complete the process. At this point you can return to the “Manage Configurations” form to review the configurations you just added:



In order to allow the *SmooksTransformer* JBoss ESB ActionProcessor to leverage these configurations, simply make sure the *smooks-cdr.lst* file (in the root of the classpath) contains a list entry of **http://<host>:<port>/jboss-esb-console/transform/smooks-config.jsp**. where host and port refer to the address of the running Admin Conosle application.

An *SmooksTransformer* action configuration to execute these resources on this message flow would appear in your ESB Listener Configuration as follows:

```
<action class="org.jboss.soa.esb.actions.converters.SmooksTransformer"
        from-type="text/csv:fullFillOrder"
        from="DVDStore:OrderDispatchService"
        to-type="text/xml:recordOrder"
        to="DataWarehouse:OrderTrackingService"
        />
```

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*.