Smooks Example - db-edi-etl

This example illustrates how Smooks can be used to extract data from an EDI message and load this data into a database, all without writing a single line of code (except for the "Main" class that executes Smooks).

- Overview
- The Input Message
- Binding the Order Data into the Virtual Data Model
- Configuring The Database Datasource
- Configuring The SQLExecutor Resources
- Executing Smooks

SVN - **Download** - **Other Tutorials**

To Build: "mvn clean install"
To Run: "mvn exec:java"

Overview

This example works by filtering the sample EDI message defined in input-message.edi using the EdiSax parser configuration defined in edi-to-sax-order-mapping.xml. As it filters the message, Smooks extracts and binds components from the message into a Virtual Data Model (defined in smooks-configs/bindings.xml). This binding data is then used in a number of SQLExecutor resources which are triggered to execute (perform inserts on the database) on passing each order and order item in the message.

An important feature of this example is how it manages its memory footprint. It processes the message using the SAX filter (therefore no in memory DOM), persisting the order and order items without ever holding more than the current order's header info, plus the current order-item being processed i.e. it never holds a full order in memory. This means that Smooks can process huge messages (GBs) using this processing model. The same principals can easily applied to splitting, transforming and routing of huge messages.

The Input Message

The sample input EDI message is as follows:

```
MLS*Wed Nov 15 13:45:28 EST 2006
HDR*1*0*59.97*64.92*4.95
CUS*user1*Harry^Fletcher*SD
ORD*1*1*364*The 40-Year-Old Virgin*28.98
ORD*2*1*299*Pulp Fiction*30.99
HDR*2*0*81.30*91.06*9.76
CUS*user2*George^Hook*SD
ORD*3*2*983*Gone with The Wind*25.80
ORD*4*3*299*Lethal Weapon 2*55.50
```

To convert this EDI message stream format into a stream of SAX events that can be processed by Smooks, we use the EdiSax parser with the following mapping configuration (defined in edi-to-sax-order-mapping.xml).

```
<?xml version="1.0" encoding="UTF-8"?>
<medi:edimap
xmlns:medi="http://www.milyn.org/schema/edi-message-mapping-1.0.xsd">

    <medi:description name="DVD Order" version="1.0"/>

    <medi:delimiters segment="&#10;" field="*" component="^"
sub-component="~"/>

    <medi:segments xmltag="orders">

        <medi:segment segcode="MLS" xmltag="message-header">
            <medi:field xmltag="date"/>
        </medi:segment>

        <medi:segment segcode="HDR" xmltag="order" minOccurs="1"
maxOccurs="-1">
            <medi:field xmltag="order-id"/>
            <medi:field xmltag="status-code"/>
            <medi:field xmltag="net-amount"/>
            <medi:field xmltag="total-amount"/>
            <medi:field xmltag="tax"/>

            <medi:segment segcode="CUS" xmltag="customer-details"
minOccurs="1" maxOccurs="1">
                <medi:field xmltag="username"/>
                <medi:field xmltag="name">
                    <medi:component xmltag="firstname"/>
                    <medi:component xmltag="lastname"/>
                </medi:field>
                <medi:field xmltag="state"/>
            </medi:segment>

            <medi:segment segcode="ORD" xmltag="order-item" minOccurs="1"
maxOccurs="-1">
                <medi:field xmltag="position"/>
                <medi:field xmltag="quantity"/>
                <medi:field xmltag="productId"/>
                <medi:field xmltag="title"/>
                <medi:field xmltag="price"/>
            </medi:segment>

        </medi:segment>

    </medi:segments>

</medi:edimap>
```

We configure the EDI parser as follows (defined in smooks-configs/edi-orders-parser.xml):

```
<?xml version="1.0"?>
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.0.xsd">

    <!--
    Configure the EDI Parser to parse the message stream into a stream of
SAX events.
    -->
    <resource-config selector="org.xml.sax.driver">
        <resource>org.milyn.smooks.edi.SmooksEDIParser</resource>
        <param name="mapping-model">/edi-to-sax-order-mapping.xml</param>
    </resource-config>

</smooks-resource-list>
```

Binding the Order Data into the Virtual Data Model

In order to process the order data contained in the EDI message, we extract items from the message stream as it is being filtered and bind them into a Virtual Data Model (Vs defining a physical Java Data Model). The process of doing this is the same as for any of the other samples that use the Javabean Cartridge (see *-to-java tutorials).

The binding configuration is as follows (defined in smooks-configs/bindings.xml):

```
<?xml version="1.0"?>
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.0.xsd">

    <!--
        Virtual Model Binding Configurations for the order message
elements...

        Just capturing the order and order-item element details into 2
        Maps, overwriting each as we iterate through the message i.e. not
        accumulating them in memory => low memory footprint because we only
        have details of the current order + current order-item in memory
        at any given time (i.e. we never even have a full order in
memory)...

        The Database Schema (need to capture enough from the message to
populate this):
        ORDERS (ORDERNUMBER INTEGER, USERNAME VARCHAR(50), STATUS INTEGER,
NET DOUBLE, TOTAL DOUBLE, ORDDATE DATE)
        ORDERITEMS (ORDERNUMBER INTEGER, QUANTITY INTEGER, PRODUCT INTEGER,
TITLE VARCHAR(50), PRICE DOUBLE)
    -->

    <resource-config selector="message-header">
        <resource>org.milyn.javabean.BeanPopulator</resource>
        <param name="beanId">message</param>
        <param name="beanClass">java.util.HashMap</param>
        <param name="bindings">
            <binding property="date" selector="message-header date"
type="OrderDateDecoder" /> <!-- Defined below -->
```

```
        </param>
    </resource-config>

    <resource-config selector="order">
        <resource>org.milyn.javabean.BeanPopulator</resource>
        <param name="beanId">order</param>
        <param name="beanClass">java.util.HashMap</param>
        <param name="bindings">
            <binding property="orderNum"      selector="order order-id"
type="Integer" />
            <binding property="customerUname" selector="order
customer-details username" />
            <binding property="status"        selector="order status-code"
type="Integer" />
            <binding property="net"           selector="order net-amount"
type="BigDecimal" />
            <binding property="total"         selector="order total-amount"
type="BigDecimal" />
        </param>
    </resource-config>

    <resource-config selector="order-item">
        <resource>org.milyn.javabean.BeanPopulator</resource>
        <param name="beanId">orderItem</param>
        <param name="beanClass">java.util.HashMap</param>
        <param name="bindings">
            <!-- Just bind in all elements of the orderItem into the
orderItem map.  Property name is
                 taken from the element name... -->
            <binding selector="order-item *" />
        </param>
    </resource-config>

    <resource-config selector="decoder:OrderDateDecoder">
        <resource>org.milyn.javabean.decoders.DateDecoder</resource>
        <param name="format">EEE MMM dd HH:mm:ss z yyyy</param>
    </resource-config>
```

```
</smooks-resource-list>
```

Configuring The Database Datasource

The inserts are performed on the database via the SQLExecutor element visitor. This resource requires a database datasource resource to be configured in Smooks. Smooks supports 2 types of datasource resources:

1. DirectDataSource: A direct connection datasource, where the driver etc is explicitly defined in the resource configuration.
2. JndiDataSource: A javax.sql.DataSource resource accessed via the local JNDI tree.

In this example, we use a Hypersonic database and for it we configure a DirectDataSource as follows (definde in smooks-configs/datasources.xml):

```
<?xml version="1.0"?>
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.0.xsd">

    <resource-config selector="$document">
        <resource>org.milyn.db.datasource.DirectDataSource</resource>
        <param name="datasource">DBExtractTransformLoadDS</param>
        <param name="driver">org.hsqldb.jdbcDriver</param>
        <param
name="url">jdbc:hsqldb:hsql://localhost:9201/milyn-hsql-9201</param>
        <param name="username">sa</param>
        <param name="password"></param>
        <param name="autoCommit">false</param>
    </resource-config>

</smooks-resource-list>
```

See the database creation script in db-create.script in the example source.

Configuring The SQLExecutor Resources

The top level configuration in this sample pulls together the resource configs for the EDI parser, Data Bindings and Database Datasource. It then adds the SQLExecutor configurations that use the data bindings via the DBExtractTransformLoadDS Datasource.

The configuration is as follows (smooks-configs/smooks-config.xml):

```
<?xml version="1.0"?>
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.0.xsd">

    <!--
    Filter the message using the SAX Filter (i.e. not DOM, so no
    intermediate DOM, so we can process huge messages...
    -->
    <resource-config selector="global-parameters">
        <param name="stream.filter.type">SAX</param>
    </resource-config>

    <!--
    Define the EDI stream parser for the orders message...
```

```
    -->
    <import file="edi-orders-parser.xml" />

    <!--
    Define the Database Datasource(s)...
    -->
    <import file="datasources.xml"/>

    <!--
    Define the Data Bindings.  This is to bind the order and orderItem data
into the ExecutionContext so it
    can be used by the SQLExecutor for performing the inserts...
    -->
    <import file="bindings.xml"/>

    <!--
========================================================================
========
    Now define the SQLExecutor resource that will use the data bound from
the EDI message
    into the virtual data model defined in bindings.xml...

========================================================================
========= -->

    <!-- Assert whether it's an insert or update.  Need to do this just
before we do the
         insert/update, which is triggered to happen just after the
customer-details are processed... -->
    <resource-config selector="customer-details">
        <resource>org.milyn.db.SQLExecutor</resource>
        <param name="executeBefore">true</param>
        <param name="datasource">DBExtractTransformLoadDS</param>
        <param name="statement">select ORDERNUMBER from ORDERS where
ORDERNUMBER = ${order.orderNum}</param>
        <param name="resultSetName">orderExistsRS</param>
    </resource-config>

    <!-- If it's an insert (orderExistsRS.isEmpty()), insert the order at
the end of the customer-details i.e. just before we process the order
items... -->
    <resource-config selector="customer-details">
        <resource>org.milyn.db.SQLExecutor</resource>
        <condition
evaluator="org.milyn.javabean.expression.MVELExpressionEvaluator">orderExi
stsRS.isEmpty()</condition>
        <param name="executeBefore">false</param>
        <param name="datasource">DBExtractTransformLoadDS</param>
        <param name="statement">INSERT INTO ORDERS
VALUES(${order.orderNum}, ${order.customerUname}, ${order.status},
${order.net}, ${order.total}, ${message.date})</param>
    </resource-config>
```

```
    <!-- And insert each orderItem... -->
    <resource-config selector="order-item">
        <resource>org.milyn.db.SQLExecutor</resource>
        <condition
evaluator="org.milyn.javabean.expression.MVELExpressionEvaluator">orderExi
stsRS.isEmpty()</condition>
        <param name="executeBefore">false</param>
        <param name="datasource">DBExtractTransformLoadDS</param>
        <param name="statement">INSERT INTO ORDERITEMS VALUES
(${order.orderNum}, ${orderItem.quantity}, ${orderItem.productId},
${orderItem.title}, ${orderItem.price})</param>
    </resource-config>

    <!-- Ignoring updates for now!! -->
```

```
</smooks-resource-list>
```

Executing Smooks

See the example/Main.java in the example source.