# Smooks Example - model-driven-basic

The primary goal of this example is to illustrate how Smooks can be used to perform a Model Driven Transformation using the Smooks Javabean and Templating Cartridges.



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

**NOTE**: Take a look the at "java-basic" tutorial before reading this tutorial.

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

## The Input and Output Messages

The input to the transformation is Shipping History message as follows:

```
<history warehouse="2">
    <header>
        <date>2007-07-16T19:20:30</date>
    </header>
    <trackingNumbers>
        Speedy:ZX5-20030294-731
        Acme:Previous0987
        Zoom:776-ASDAAAA-988
    </trackingNumbers>
</history>
```

As you can see from this message, the <trackingNumbers> element contains the tracking numbers, but in a format that is not easily consumed.

The above input message needs to be transformed into the following output message:

```
<shipping-history date="2007-07-16">
    <warehouse id="2" location="Belfast" />
    <trackingNumbers>
        <trackingNumber>
            <shipperID>Speedy</shipperID>
            <shipmentNumber>ZX5-20030294-731</shipmentNumber>
        </trackingNumber>
        <trackingNumber>
            <shipperID>Acme</shipperID>
            <shipmentNumber>Previous0987</shipmentNumber>
        </trackingNumber>
        <trackingNumber>
            <shipperID>Zoom</shipperID>
            <shipmentNumber>776-ASDAAAA-988</shipmentNumber>
        </trackingNumber>
    </trackingNumbers>
</shipping-history>
```

We will transform this message by first using it to populate a Java Object Model and then from that model, we generate the required output using a FreeMarker template.

# The Object Model

We have 3 basic types - ShippingHistory , Warehouse and TrackingNumber.

The ShippingHistory (getters and setters not shown) contains a Warehouse and a TrackingNumber list:

```
public class ShippingHistory {
    private Warehouse warehouse;
    private Date creationDate;
    private TrackingNumber[] trackingNumbers;
}
```

The Warehouse (getters and setters not shown):

```
public class Warehouse {
    private int id = -1;
}
```

Note from the Warehouse Java source that this class also supports a "getName" method which returns a Warehouse name, which is looked up from a local hash. This is basic form of model enrichment.

The TrackingNumber (getters and setters not shown):

```
public class TrackingNumber {
    private String shipperID;
    private String shipmentNumber;
}
```

# Model Population

The model is populated (from the input message) using the <jb:binding> configurations of the Javabean Cartridge.

The configuration to populate the model is as follows. See the <jb:bindings> configurations (also shows the FreeMarker templating resource):

```
<?xml version="1.0"?>
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.1.xsd"

xmlns:jb="http://www.milyn.org/xsd/smooks/javabean-1.1.xsd"

xmlns:ftl="http://www.milyn.org/xsd/smooks/freemarker-1.1.xsd">

    <!--
    Use a FreeMarker template to perform the model driven transformation.
You can also inline the template here.
    -->
    <ftl:freemarker applyOnElement="$document">
        <ftl:template>/example/templates/HistoryTrans.ftl</ftl:template>
    </ftl:freemarker>

    <!--
    Configure the History bean creation and population.
    -->
    <jb:bindings beanId="history" class="example.model.ShippingHistory"
createOnElement="history">
        <jb:wiring property="warehouse" beanIdRef="warehouse"/>
        <!-- Note the date decoder is not configured with a format.  Will
therefore use the default (SOAP).  See DateDecoder class. -->
        <jb:value property="creationDate" decoder="Date" data="header/date"
/>
        <!-- Note the "special" decoder for the trackingNumbers. -->
        <jb:value property="trackingNumbers"
decoder="example.model.TrackingNumberDecoder"
data="history/trackingNumbers" />
    </jb:bindings>

    <!--
    Configure the Warehouse bean creation and population.
    Note how we create it on visiting the history element and set the bean
on
    history element (will set it based on it's own beanId).
    -->
    <jb:bindings beanId="warehouse" class="example.model.Warehouse"
createOnElement="history">
        <jb:value property="id" decoder="Integer" data="history/@warehouse"
/>
        <!-- Note the "special" decoder for mapping the warehouse id. -->
        <jb:value property="name" decoder="Mapping"
data="history/@warehouse">
            <jb:decodeParam name="1">Dublin</jb:decodeParam>
            <jb:decodeParam name="2">Belfast</jb:decodeParam>
            <jb:decodeParam name="3">Cork</jb:decodeParam>
        </jb:value>
    </jb:bindings>

</smooks-resource-list>
```

So we have <jb:binding> configurations, one for the ShippingHistory instance and one for the Warehouse instance. Because of the nature of the <trackingNumbers> element, where all the tracking records are bundled into a single text node (line separated), we need to write a "special" Data Decoder class for decoding this element into a list of TrackingNumber instances. This decoder is implemented in the TrackingNumberDecoder cla ss.

For more info on Javabean decoders, see the DataDecoder Javadocs.

# FreeMarker Template Application

Once the Object Model has been populated, we want Smooks to apply the FreeMarker template to the model to produce the required output message (shown above).

The template is configured in Smooks as follows (extract from smooks-config.xml):

```
<!--
    Use a FreeMarker template to perform the model driven transformation.
You can also inline the template here.
    -->
    <ftl:freemarker applyOnElement="$document">
        <ftl:template>/example/templates/HistoryTrans.ftl</ftl:template>
    </ftl:freemarker>
```

And the actual template can be seen here. Note how concise the template is.

# Executing Smooks

As with the java-basic tutorial, we use the Smooks class as follows:

```
Smooks smooks = new Smooks("smooks-config.xml");
StringResult result = new StringResult();

smooks.filter(new StreamSource(messageIn), result);

// Transformation result available in outputWriter ...
```

Of course, you'd typically cache the Smooks instance.

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