XML Transformation using the Javabean and Templating Cartridges

Duration: 5 Minutes.

This Tutorial will illustrate how the JavaBean and Templating Smooks Cartridges can be used to perform template based transformations.

This sample is based on a tutorial from webservices.xml.com. The tutorial in question actually demonstrates 2 scenarios where XSLT can be used for webservice integration. The 1st uses XSLT only and the 2nd uses XSLT plus Java (within a Servlet). The second scenario encorporates Java because of the difficulty of performing parts of the transformation with XSLT alone (go to the end of the page).

This sample will illustrate how the second transformation scenario can be accomplished using the Smooks JavaBean and Templating Cartridges. It will demonstrate how Smooks offers different options when it comes to templating (using either the XSLT or StringTemplate templating solutions) as well as making it easy to implement the more complex logic in Java (via the JavaBean Cartridge).

Transformation Requirements

Basically, the SOAP request needs to be transformed from (not including the full SOAP message):

```
<trackingNumber>
 <shipperID>Speedy</shipperID>
 <shipmentNumber>
  ZX5-20030294-731
 </shipmentNumber>
</trackingNumber>
```

To (not including the full SOAP message):

```
<trackingNumber>
 Speedy:ZX5-20030294-731
</trackingNumber>
```

And the response needs to be transformed from (not including the full SOAP message):

```
<history>
  Speedy:ZX5-20030294-731
  Acme:Previous0987
</history>
```

To (not including the full SOAP message):

```
<trackingNumbers>
 <trackingNumber>
  <shipperID>Speedy</shipperID>
  <shipmentNumber>
   ZX5-20030294-731
  </shipmentNumber>
 </trackingNumber>
 <trackingNumber>
  <shipperID>Acme</shipperID>
  <shipmentNumber>
   Previous0987
  </shipmentNumber>
 </trackingNumber>
</trackingNumbers>
```

Write JavaBean Classes

We need 2 JavaBean classes. We will implement the **TrackingNumber** bean for storing data extracted from the incoming XML request and we will implement the **History** bean to store data extracted from the response. These beans will then be used by the XSLT/StringTemplate templates to perform the transformations.

On the **TrackingNumber** bean, note how we use the toString method.

```
public class TrackingNumber {

    private String shipperID;
    private String shipmentNumber;

    public String getShipperID() {
        return shipperID;
    }
    public void setShipperID(String shipperID) {
        this.shipperID = shipperID.trim();
    }
    public String getShipmentNumber() {
        return shipmentNumber;
    }
    public void setShipmentNumber(String shipmentNumber) {
        this.shipmentNumber = shipmentNumber.trim();
    }

    public String toString() {
        return shipperID + ":" + shipmentNumber;
    }
}
```

Note how the **History** bean *setTrackingNumbers* method (below) parses the tracking-number history list from the endpoint response to produce a list of the **TrackingNumber** bean instances that can later be retrieved from the **History** bean via the *getTrackingNumbers* method to help perform the endpoint response transformation. So, we're reusing the **TrackingNumber** bean - it will actually be used in both transformations (request and response).

```
public class History {

    private TrackingNumber[] trackingNumbers;
    private static Pattern lineSplitter =
        Pattern.compile("$", Pattern.MULTILINE);

    public void setTrackingNumbers(String historyText) {
        // break the history up line by line - 1 tracking-number per line
        String[] unparsedTrackingNumber = lineSplitter.split(historyText);
        Vector tnList = new Vector(unparsedTrackingNumber.length);

        // iterate over and parse the tracking-number lines
        for (int i = 0; i < unparsedTrackingNumber.length; i++) {
            String[] tokens = unparsedTrackingNumber[i].trim().split(":");

            if(tokens.length == 2) {
                TrackingNumber trackingNumber = new TrackingNumber();

                trackingNumber.setShipperID(tokens[0]);
                trackingNumber.setShipmentNumber(tokens[1]);
                tnList.add(trackingNumber);
            }
        }

        trackingNumbers = new TrackingNumber[tnList.size()];
        tnList.toArray(trackingNumbers);
    }

    public TrackingNumber[] getTrackingNumbers() {
        return trackingNumbers;
    }
}
```

Write the Templates (XSLT or StringTemplate)

- XSLT
- StringTemplate

Note, it's worth comparing both solutions here!

Write the Resource Targeting Configurations

- XSLT
- StringTemplate

Note again, it's worth comparing both solutions here! Note in particular how the JavaBean configurations are exactly the same in both solutions.

Execute the Transformations

```
SmooksStandalone smooks = new SmooksStandalone("UTF-8");

// Configure Smooks - register the request and response useragents...
smooks.registerUseragent("shipping-request");
smooks.registerUseragent("shipping-response");
// Register the templating CDU creators.  Basically, these configs tell
Smooks how to handle the .st files.
TemplatingUtils.registerCDUCreators(smooks.getContext());
// Register the .cdrl configurations...
smooks.registerResources("trans-request",
getClass().getResourceAsStream("trans-request.cdrl"));
smooks.registerResources("trans-response",
getClass().getResourceAsStream("trans-response.cdrl"));

// Transform and serialise the request...
Node requestTrans = smooks.filter("shipping-request", requestStream);
smooks.serialize("shipping-request", requestTrans, requestOutputWriter);

// ... transformed request submitted to target endpoint...

// Transform and serialise the endpoint response...
Node responseTrans = smooks.filter("shipping-response", responseStream);
smooks.serialize("shipping-response", responseTrans, responseOutputWriter);
```

Note: This sample is implemented as a set of unit tests that can be viewed through FishEye. The parts of the tests specific to XSLT and StringTemplate are located in the "xslt" and "st" folders respectively.