TransformationFeature

Transformation Feature

The CXF Transformation feature provides for a flexible and fast way to do dynamic transformation of inbound and/or outbound XML messages.

This feature can be used in a number of cases: dropping the namespace of the outbound messages, qualifying the incoming message, changing namespaces, appending or dropping elements and converting attributes to elements.

The "outTransformElements", "inTransformElements", "outDropElements", "inDropElements", "outAppendElements", "inAppendElements" and "attributesAsElements" properties can be used.

On the outbound service side, some of the transformation features (e.g. "outTransformElements", "outDropElements") may not work properly prior to CXF versions 3.2.6 and 3.1.17. This is due to outputstream optimization that CXF does. In this case, the outputstream optimization must be explicitly disabled for the outbound transformation feature to work properly by setting the JAX-WS/JAX-RS property "disable.outputstream.optimization" to "true".

Spring configuration

Changing input and output element names and namespaces

"outTransformElements" map property can be used to change the output element names and change or drop namespaces. Keys are the elements to be changed, values are the new element names. Example:

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="outTransformElements">
    <map>
      <!-- change "book" to "thebook" -->
      <entry key="book" value="thebook"/>

      <!-- drop the namespace from "book" -->
      <entry key="{http://books}book" value="book"/>

      <!-- qualify "book" with "http://books" -->
      <entry key="book" value="{http://books}thebook"/>

      <!--  change namespace to "http://books" for all the elements with the "http://book" namespace -->
      <entry key="{http://book}*" value="{http://books}*"/>
    </map>
  </property>
</bean>
```

"inTransformElements" map property can be used to change the input element names and change or drop namespaces. See the "outTransfromElements" property description for an example.

Appending new input and output elements

"outAppendElements" and "inAppendElements" map properties can be used to append new simple or qualified elements to the output/input in a number of ways. Keys are the elements the new elements will be appended before, values are the new elements. Examples:

Append-Pre-Wrap

Using inAppendsElements:

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="inAppendElements">
    <map>
      <!-- get "book" wrapped with the new "thebook" element-->
      <entry key="book" value="thebook"/>
    </map>
  </property>
</bean>
```

Using outAppendsElements:

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="outAppendElements">
    <map>
      <!-- get "book" wrapped with the new "thebook" element-->
      <entry key="book" value="thebook"/>
    </map>
  </property>
</bean>
```

Append-Post-Wrap

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="inAppendElements">
    <map>
      <!--
         get all "book" children elements wrapped with the new "thebook" element
         using the "/" convention.
      -->
      <entry key="book/" value="thebook"/>
    </map>
  </property>
</bean>
```

Append-Pre-Include

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="inAppendElements">
    <map>
      <!-- append new simple "thebook" element with a text value '2' before the "book" element -->
      <entry key="book" value="thebook=2"/>
    </map>
  </property>
</bean>
```

Append-Post-Include

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="inAppendElements">
    <map>
      <!-- append new simple "thebook" element with a text value '2' using the "/" convention as the last child
element within the "book" element -->
      <entry key="book/" value="thebook=2"/>
    </map>
  </property>
</bean>
```

Comparing four append modes

input	append-pre-wrap	append-post-wrap	append-pre-include	append-post-include
	key="book" value="thebook"	key="book/" value="thebook"	key="book" value="thebook=2"	key="book/" value="thebook=2"
<sales> <book> <title>CXF ... </title> <price>38.68</price> </book> </sales>	<sales> <thebook> <book> <title>CXF ... </title> <price>38.68</price> </book> </thebook> </sales>	<sales> <book> <thebook> <title>CXF ... </title> <price>38.68</price> </thebook> </book> </sales>	<sales> <thebook>2</thebook> <book> <title>CXF ... </title> <price>38.68</price> </book> </sales>	<sales> <book> <title>CXF ... </title> <price>38.68</price> <thebook>2</thebook> </book> </sales>

Replacing text content

It's possible to replace the text content of a given simple element only on the input and output, for example:

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="inAppendElements">
    <map>
      <!-- replace the text content of {ns}a element with the 'new Text' value -->
      <entry key="{ns}a" value="{ns}a=new Text"/>
    </map
  </property>
</bean>
```

Dropping output and input elements

"outDropElements" and "inDropElements" list properties can be used to drop output and input elements. Note that children elements if any of a given dropped element are not affected. Please see the "outDropElements" property description for an example. It's a so-called "shallow" drop.

Additionally, outTransformElements and inTransformElements property can be used to deep-drop an element and all of its children if any, for example:

```
<bean id="transformFeature" class="org.apache.cxf.feature.StaxTransformFeature">
  <property name="outTransformElements">
    <map>
      <!-- drop "book" and all of its children, using an empty value convention -->
      <entry key="book" value=""/>

    </map>
  </property>
</bean>
```

Converting attributes to elements

"attributesAsElements" boolean property can be used to have attributes serialized as elements on the output only.

The combination of "attributesAsElements" and "outDropElements" properties can be used to have certain attributes ignored in the output by turning them into elements and then blocking them.

Input Transformation and Redirection

Consider the case where a new endpoint has been introduced but some of the existing clients have not been updated yet to work with the new endpoint, they are still unaware of it.

In this case you may want to keep the CXFServlet serving the old clients but make it redirect them to a new CXFServlet serving a new endpoint only. Now, in order to serve the old clients one needs to apply a transform feature, however the new clients should not be affected. Thus the feature can be configured such that it's only triggered if a certain contextual property has been set on a current Message. In this case the feature should only apply to the old redirected clients:

```
<bean id="transformFeatureRest" class="org.apache.cxf.feature.StaxTransformFeature">
      <!--
         apply the transformation only if the boolean property with the given name
         is set to true on the message
      -->
      <property name="contextPropertyName" value="http.service.redirection"/>
      <!-- the transform configuration -->
</bean>
```

Configuring the feature from the code

The feature can be configured from the code for JAX-WS or JAX-RS clients and endpoints.

JAX-WS

Here is how a JAX-WS client can be configured:

```
CustomerServiceService service = new CustomerServiceService();
CustomerService customerService = service.getCustomerServicePort();
Client client = ClientProxy.getClient(customerService);

// drop namespace from all elements qualified by 'http://customers'
Map<String, String> outTransformMap = Collections.singletonMap("{http://customers}*", "*");
org.apache.cxf.interceptor.transform.TransformOutInterceptor transformOutInterceptor =
    new org.apache.cxf.interceptor.transform.TransformOutInterceptor();
transformOutInterceptor.setOutTransformElements(outTransformMap);
client.getOutInterceptors().add(transformOutInterceptor);

// qualify the incoming 'customer' element with 'http://customers'
Map<String, String> inTransformMap =  Collections.singletonMap("customer", "{http://customers}customer");
org.apache.cxf.interceptor.transform.TransformInInterceptor transformInInterceptor =
    new org.apache.cxf.interceptor.transform.TransformInInterceptor();
transformInInterceptor.setInTransformElements(inTransformMap);
client.getInInterceptors().add(transformInInterceptor);
```

JAX-RS

Here is how a JAX-RS client can be configured:

```
CustomerService customerServiceProxy = JAXRSClientFactory.create(endpointAddress, CustomerService.class);

ClientConfiguration config = WebClient.getConfig(customerServiceProxy);

// or
//WebClient client = WebClient.create(endpointAddress);
//ClientConfiguration config = WebClient.getConfig(client);

// drop namespace from all elements qualified by 'http://customers'
Map<String, String> outTransformMap = Collections.singletonMap("{http://customers}*", "*");
org.apache.cxf.interceptor.transform.TransformOutInterceptor transformOutInterceptor =
    new org.apache.cxf.interceptor.transform.TransformOutInterceptor();
transformOutInterceptor.setOutTransformElements(outTransformMap);
config.getOutInterceptors().add(transformOutInterceptor);

// qualify the incoming 'customer' element with 'http://customers'
Map<String, String> inTransformMap =  Collections.singletonMap("customer", "{http://customers}customer");
org.apache.cxf.interceptor.transform.TransformInInterceptor transformInInterceptor =
    new org.apache.cxf.interceptor.transform.TransformInInterceptor();
transformInInterceptor.setInTransformElements(inTransformMap);
config.getInInterceptors().add(transformInInterceptor);
```

Transform interceptors and phases

TransformInInterceptor and TransformOutInterceptor interceptors run in POST_STREAM and PRE_STREAM phases.
In some cases it may be needed to change the phase, for example, the in transformation has to be applied after the encrypted payload has been decrypted and its signature checked. For such transformations to succeed TransformInInterceptor will need to be created with a constructor accepting a 'phase' String parameter, for the interceptor to run after the decrypption and signature validation actions have been performed.

Default namespace on the output

The 'outDefaultNamespace' feature property can be used to enforce the default namespace declaration. The value of this property has to match one of the out namespaces.