Smooks Standalone Tutorial

This tutorial illustrates how SmooksXML can be used for XML transformation outside a container via SmooksStandalone.

So, we'll base this tuorial on the Smooks In a Servlet Container turorial:

1. For "message-target1", we will convert <aaa> elements to <zzz> elements.
2. For "message-target1", we will convert <ccc> elements to <xxx> elements.
3. For "message-target2", we will convert <aaa> elements to <zzz> elements.

So, when transforming the message "<aaa><bbb>888</bbb><ccc>999</ccc></aaa>", we want:

1. **message-target1** to receive "<zzz><bbb>888</bbb><xxx>999</xxx></zzz>", and
2. **message-target2** to receive "<zzz><bbb>888</bbb><ccc>999</ccc></zzz>".

Write the Transformation Unit

So, the transformation code to perform this task is very simple:

```
public class RenameElementTrans extends AbstractProcessingUnit {

 // cache the new element name.
 private String newElementName;

 public RenameElementTrans(SmooksResourceConfiguration config) {
  super(config);
  // Capture the new name for the element from the configuration...
  newElementName = config.getStringParameter("new-name");
 }

 public void visit(Element element, ContainerRequest containerRequest) {
  // Rename the element to the configured new name.
  DomUtils.renameElement(element, newElementName, true, true);
 }

 public boolean visitBefore() {
  // We want this transformation code to visit the target element after all
  // child elements have been iterated over.
  return false;
 }
}
```

The constructor and "visit" method code is of most interest. In the constructor we capture a parameter from the transformation configuration (talked about below). In the "visit" method, we use this parameter value to perform the rename. So as you can see, we've actually implemented the transformation unit such that it can be reused for renaming any element.

Classes of interest here are:

- AbstractProcessingUnit
- SmooksResourceConfiguration
- ContainerRequest
- DomUtils

Target the Transformation Unit

As you'll have already noticed, **message-target1** and **message-target2** transformations have something in common i.e. they both need <aaa> elements converted to <zzz> elements. We can use profiling to target this transformation at both message targets i.e. make both message targets members of the same profile. Using profiles here is a little silly, but it still makes the illustration.

The first piece of code creates the SmooksStandalone instance and initialises it with the useragents (and their profiles). This basically tells the SmooksStandalone instance that it will be transforming messages for these targets.

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

// Add the 2 useragents and configure them with profiles - note "profile2"
is common to both...
smooks.registerUseragent("message-target1", new String[] {"profile1",
"profile2"});
smooks.registerUseragent("message-target2", new String[] {"profile2",
"profile3"});
```

Next we create the resource configuration file and add it somewhere in the application classpath. This configuration tell the SmooksStandalone instance when to trigger the above RenameElementTrans ProcessingUnit i.e. on which target elements, and for which message targets. Note we can use the useragent name or a profile name for useragent targeting. Profiles simply allow us to reduce the number of configuration entities where two or more useragent targets require the same transformation.

```
<?xml version="1.0"?>
<!DOCTYPE smooks-resource-list PUBLIC "-//MILYN//DTD SMOOKS 1.0//EN"
"http://milyn.codehaus.org/dtd/smooksres-list-1.0.dtd">

<smooks-resource-list default-path="com.acme.trans.RenameElementTrans">
 <smooks-resource useragent="message-target1" selector="ccc">
  <param name="new-name">xxx</param>
 </smooks-resource>
 <smooks-resource useragent="profile2" selector="aaa">
  <param name="new-name">zzz</param>
 </smooks-resource>
</smooks-resource-list>
```

The resource configuration is registered with the SmooksStandalone instance as follows:

```
smooks.registerResources(getClass().getResourceAsStream("<file-classpath-l
ocation>"));
```

Alternatively, we could have registered these resource configurations by constructing a set of SmooksResourceConfiguration instances, calling the registerResource method on the SmooksStandalone instance for each SmooksResourceConfiguration instances.

Execute the Transformation

Now, to transform a message stream for one of these message targets:

```
Node transResult = smooks.filter("message-target1", messageInputStream);
```

Or, if transforming a DOM message:

```
Node transResult = smooks.filter("message-target1", messageDocument);
```