Sample usage of umsCodeGenerator

Table of contents

1. Sample Files in the distribution

In 'sample' directory in the distribution, there are six **mapping definition files** for umsCodeGenerator as well as test program, input data and expected result to be combined with these mapping files. Correspondence between **mapping definition files**, test programs, input data and expected results are shown in the following table.

Input Data	mapping definition	Test Program	Language	Expected Result
(a')	(B)	(c)		
textSampleTest.dat (sample/textSampleTest.dat)	textSample.dat-perl (sample/textSample.dat-perl)	textSampleTest.pl (sample/textSampleTest.pl)	PERL	textSampleTest.result (sample/textSampleTest.result)
textSampleTest.dat (sample/textSampleTest.dat)	textSample.dat-clng (sample/textSample.dat-clng)	textSampleTest.c (sample/textSampleTest.c)	C	textSampleTest.result (sample/textSampleTest.result)
textSampleTest.dat (sample/textSampleTest.dat)	textSample.dat-java (sample/textSample.dat-java)	textSampleTest.java (sample/textSampleTest.java)	C	textSampleTest.result (sample/textSampleTest.result)
textSampleTest.dat (sample/textSampleTest.dat)	fixedSample.dat-perl (sample/fixedSample.dat-perl)	fixedSampleTest.pl (sample/fixedSampleTest.pl)	PERL	textSampleTest.result (sample/textSampleTest.result)
textSampleTest.dat (sample/textSampleTest.dat)	(N/A)	(N/A)	C	textSampleTest.result (sample/textSampleTest.result)
textSampleTest.dat (sample/textSampleTest.dat)	(N/A)	(N/A)	JAVA	textSampleTest.result (sample/textSampleTest.result)
(none)	xrteStatus.dat-perl (sample/xrteStatus.dat-perl)	xrteStatusTest.pl (sample/xrteStatusTest.pl)	PERL	xrteStatusTest.result (sample/xrteStatusTest.result)
(none)	xrteStatus.dat-clng (sample/xrteStatus.dat-clng)	xrteStatusTest.c (sample/xrteStatusTest.c)	C	xrteStatusTest.result (sample/xrteStatusTest.result)
(none)	(N/A)	(N/A)	JAVA	xrteStatusTest.result (sample/xrteStatusTest.result)

2. Step 1. Extraction of input schema and output schema

A mapping definition file can be created from **schema of input** and **schema of output**. On the other hand, **mapping definition file** can be split into **schema of input (description of external data)** and **schema of output (description of interface with program language)**. Spliting of **mapping definition file** in XML syntax can be performed with style sheet of XSLT. Current version of umsCodeGenerator deals with **one input space** and **three output**

spaces. There are four (**one** + **three**) XSLT style sheets in the distribution.

- selectDat.xsl (classes/selectDat.xsl) : **extracts schema elements in spaces "http://ums.isas.jaxa.jp/0.3/dat" and "http://ums.isas.jaxa.jp/0.3"**
- selectPerl.xsl (classes/selectPerl.xsl) : **extracts schema elements in spaces "http://ums.isas.jaxa.jp/0.3/perl" and "http://ums.isas.jaxa.jp/0.3"**
- selectClng.xsl (classes/selectClng.xsl) : **extracts schema elements in spaces "http://ums.isas.jaxa.jp/0.3/clng" and "http://ums.isas.jaxa.jp/0.3"**
- selectJava.xsl (classes/selectJava.xsl) : **extracts schema elements in spaces "http://ums.isas.jaxa.jp/0.3/java" and "http://ums.isas.jaxa.jp/0.3"**

Sample of spliting is described in <target name="split"/> portion of 'build.xml' in the the distribution.

```
<target name="split">
  <xslt in="sample/textSample.dat-perl"  style="classes/selectDat.xsl"
out="test/textSample.dat.dat-perl" />
  <xslt in="sample/textSample.dat-clng"  style="classes/selectDat.xsl"
out="test/textSample.dat.dat-clng" />
  <xslt in="sample/textSample.dat-java"  style="classes/selectDat.xsl"
out="test/textSample.dat.dat-java" />
  <xslt in="sample/fixedSample.dat-perl"  style="classes/selectDat.xsl"
out="test/fixedSample.dat.dat-perl" />
  <xslt in="sample/xrteStatus.dat-perl"  style="classes/selectDat.xsl"
out="test/xrteStatus.dat.dat-perl" />
  <xslt in="sample/xrteStatus.dat-clng"  style="classes/selectDat.xsl"
out="test/xrteStatus.dat.dat-clng" />
  <xslt in="sample/textSample.dat-perl"  style="classes/selectPerl.xsl"
out="test/textSample.perl.dat-perl" />
  <xslt in="sample/textSample.dat-clng"  style="classes/selectClng.xsl"
out="test/textSample.clng.dat-clng" />
  <xslt in="sample/textSample.dat-java"  style="classes/selectJava.xsl"
out="test/textSample.java.dat-java" />
  <xslt in="sample/fixedSample.dat-perl"  style="classes/selectPerl.xsl"
out="test/fixedSample.perl.dat-perl" />
  <xslt in="sample/xrteStatus.dat-perl"  style="classes/selectPerl.xsl"
out="test/xrteStatus.perl.dat-perl" />
  <xslt in="sample/xrteStatus.dat-clng"  style="classes/selectClng.xsl"
out="test/xrteStatus.clng.dat-clng" />
</target>
```

Each 'xslt' element describes extraction of **input** or **output** schema. **Mapping definition file** is specified in 'in' attribute. File name of **input** or **output** schema to be generated is specified in 'out' attribute. Style sheet is specified in 'style' attribute.

This can be performed by following command.

```
ant split
```

3. Step 2. Validation of input schema and output schema

Schema of input and **schema of output** must satisfy grammar of schema of each space. When schema is described in XML, it is possible to describe some portion of grammar with XML schema languages. There are **one RELAX NG schema of input schema** and **three RELAX NG schema of output schema** in the distribution.

- ums-dat.rnc (classes/ums-dat.rnc) : **for input schema consists of spaces "http://ums.isas.jaxa.jp/0.3/dat" and "http://ums.isas.jaxa.jp/0.3"**
- ums-perl.rnc (classes/ums-perl.rnc) : **for output schema consists of spaces "http://ums.isas.jaxa.jp/0.3/perl" and "http://ums.isas.jaxa.jp/0.3"**
- ums-clng.rnc (classes/ums-clng.rnc) : **for output schema consists of spaces "http://ums.isas.jaxa.jp/0.3/clng" and "http://ums.isas.jaxa.jp/0.3"**
- ums-java.rnc (classes/ums-java.rnc) : **for output schema consists of spaces "http://ums.isas.jaxa.jp/0.3/java" and "http://ums.isas.jaxa.jp/0.3"**

Some obvious error in the description of **mapping definition** can be checked with validator of RELAX NG (eg. jing (http://www.thaiopensource.com/relaxng/jing.html)) with the schema of **input** and **output** schema.

Sample of validation is described in <target name="validate"/> portion of 'build.xml' in the distribution.

```
<target name="validate" depends="split">
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-dat.rnc"/><arg
value="test/textSample.dat.dat-perl"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-dat.rnc"/><arg
value="test/textSample.dat.dat-clng"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-dat.rnc"/><arg
value="test/textSample.dat.dat-java"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-dat.rnc"/><arg
value="test/fixedSample.dat.dat-perl"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-dat.rnc"/><arg
value="test/xrteStatus.dat.dat-perl"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-dat.rnc"/><arg
value="test/xrteStatus.dat.dat-clng"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-perl.rnc"/><arg
value="test/textSample.perl.dat-perl"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-java.rnc"/><arg
value="test/textSample.java.dat-java"/></java>
```

```
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-clng.rnc"/><arg
value="test/textSample.clng.dat-clng"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-perl.rnc"/><arg
value="test/fixedSample.perl.dat-perl"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-perl.rnc"/><arg
value="test/xrteStatus.perl.dat-perl"/></java>
  <java jar="${env.JING_HOME}/jing.jar" fork="true"><arg value="-c"/><arg
value="classes/ums-clng.rnc"/><arg
value="test/xrteStatus.clng.dat-clng"/></java>
</target>
```

This can be performed by following command.

```
ant validate
```

No cross check of this step is implemented in the following steps. If you found error in validation, you must correct mapping definition before proceed.

4. Step 3. Generation of Source Code from mapping definition file

Many programs satisfy complete behaviors described in **a mapping definition file**. You can write **such program** by yourself. Alternatively, you can generate **a few programs (one program at this moment)** with umsCodeGenerator. This step is realized by XSLT stylesheet of XALAN extended with JAVA language (http://xml.apache.org/xalan-j/extensions.html) . There are style sheets for each language. Three languages supported by method 2 of umsCodeGenerator is realized by four XSLT style sheets.

- m2perlDecoder.xsl (classes/m2perlDecoder.xsl) : stylesheet for generation of PERL program
- m2clngDecoder.xsl (classes/m2clngDecoder.xsl) : stylesheet for generation of C program
- m2chdrDecoder.xsl (classes/m2chdrDecoder.xsl) : stylesheet for generation of header file of C program
- m2javaDecoder.xsl (classes/m2javaDecoder.xsl) : stylesheet for generation of JAVA program

C language uses two style sheets, one for main body of source code and the other for header file. Other languages use one style sheet. Appropriate style sheet should be specified for generation of **each source code** .

Sample of source code generation is described in <target name="generate"/> portion of 'build.xml' in the distribution.

```
<target name="generate" depends="validate">
  <xslt in="sample/textSample.dat-perl"  style="classes/m2perlDecoder.xsl"
out="test/textSample.pl" ><classpath><pathelement/></classpath></xslt>
```

```
  <xslt in="sample/textSample.dat-clng"  style="classes/m2clngDecoder.xsl"
out="test/textSample.c" ><classpath><pathelement/></classpath></xslt>
  <xslt in="sample/textSample.dat-clng"  style="classes/m2chdrDecoder.xsl"
out="test/textSample.h" ><classpath><pathelement/></classpath></xslt>
  <xslt in="sample/textSample.dat-java"  style="classes/m2javaDecoder.xsl"
out="test/textSample.java" ><classpath><pathelement/></classpath></xslt>
  <xslt in="sample/xrteStatus.dat-perl"  style="classes/m2perlDecoder.xsl"
out="test/xrteStatus.pl" ><classpath><pathelement/></classpath></xslt>
  <xslt in="sample/xrteStatus.dat-clng"  style="classes/m2clngDecoder.xsl"
out="test/xrteStatus.c" ><classpath><pathelement/></classpath></xslt>
  <xslt in="sample/xrteStatus.dat-clng"  style="classes/m2chdrDecoder.xsl"
out="test/xrteStatus.h" ><classpath><pathelement/></classpath></xslt>
  <xslt in="sample/fixedSample.dat-perl"  style="classes/m2perlDecoder.xsl"
out="test/fixedSample.pl" ><classpath><pathelement/></classpath></xslt>
</target>
```

Each 'xslt' element describes a source code generation with style sheet. **Mapping definition file** is specified in 'in' attribute. File name of source code to be generated is specified in 'out' attribute. Stylesheet for source code generation is specified in 'style' attribute.

This can be performed by following command.

```
ant generate
```

5. Step 4. Compilation and Execution of Test Program

Test and verification of programs can be performed in the 'test' direcotry. First, you have to move current directory from the top directory with following command.

```
cd test
```

Execution of test program in compilation language (C, JAVA) requires compilation of sample and generated source code. To perform this, type following commands.

```
make compile
```

Finally, you can execute test program and compare its result with stored result. To perform this, type following commands.

```
make test
```

Compilation and execution of test above can be simply covered by following commands.

```
make
```