Schematron Testing Framework

https://github.com/MenteaXML/stf

What is Schematron?

Schematron (http://www.schematron.com/) is a language for making assertions about the presence or absence of patterns in XML documents. It is based not on grammars but on finding tree patterns in the parsed document. If you know XPath or the XSLT expression language, you can use Schematron.

Schematron allows you to develop and mix two kinds of schemas:

- assert elements allow you to confirm that the document conforms to a particular schema.
- report elements allow you to diagnose which variant of a language you are dealing with.

An example Schematron rule with one assert and one report:

```
<rule id="baz" context="baz">
  <assert role="ERROR_FOO"
          test="count(foo) = count(bar)">
Number of 'foo' and 'bar' should be equal.</assert>
  <report role="ERROR_BAR"
          test="count(bar) > 5">
'baz' should contain no more than 5 'bar'.</report>
</rule>
```

Why Use stf?

A suite of Schematron tests contains many contexts where a bug in a document will make a Schematron assert fail or a report succeed, so it follows that for any new test suite and any reasonably sized but buggy document set, there will straight away be many assert and report messages produced by the tests. When that happens, how can you be sure your Schematron tests all worked as expected? How can you separate the expected results from the unexpected? What's needed is a way to characterise the Schematron tests before you start as reporting only what they should, no more, and no less.

stf is a XProc pipeline that runs a Schematron test suite on test documents (that you create) and winnows out the expected results and report just the unexpected. stf uses a processing instruction (PI) in each of a set of (typically, small) test documents to indicate the test's expected asserts and reports: the expected results are ignored, and all you see is what's extra or missing. And when you have no more unexpected results from your test documents, you're ready to use the Schematron on your real documents.

stf was created by Mentea (http://www.mentea.net). It is Open Source and under a BSD license. You are welcome to use it, to contribute to the project, or to fork it in accordance with the terms of the license.

<?stf?>

Format of the <?stf?> processing instruction:

```
<?stf \s+ ( '#NONE' | ROLE ':' COUNT ( \s+ ROLE ':' COUNT )* ) ?>
```

stf	PI target
#NONE	No failed assert or successful report expected. Use with 'go' tests that should not produce any assert or report messages. If running Schematron on the test produces any asserts or reports, they are reported as an error.
ROLE	Token corresponding to @role value of an assert or a report in the Schematron. Schematron allows @role to be an arbitrary string, but restricting it to a single token makes it easier to deal with the PI using regular expressions rather than having to parse roles that may contain spaces.
COUNT	Integer number of expected occurrences of failed asserts or successful reports with @role value matching ROLE. A mismatch between the expected and actual count is reported as an error. A ROLE starting with # does not have its count checked.
\s	Whitespace character

<?stf?> Examples

Some sample test documents (that may be used with the Schematron rule shown above) are shown below, along with explanations of the expectations expressed by their <?stf?> processing instructions. The documents, the schema expressed by the Schematron, and the expected Schematron results are deliberately mismatched for the sake of providing the example stf output below.

```
<?stf ERROR_BAR:1 ERROR_QUX:1 #ERROR_LATER:3 ?>
<baz>
  <bar/><bar/><bar/><bar/><bar/><bar/>
</baz>
```

A failed assert or successful report with role="ERROR_BAR" is expected once in the SVRL from the test document, and either with role="ERROR_QUX" is expected once, and no assert or report with role="ERROR_LATER" is expected, since # precedes ERROR_LATER.

```
<?stf #NONE ?>
<baz>
  <bar/>
</baz>
```

No assert or report are expected for the current document.

Running stf

1. Set the properties in properties.xml to match your local setup.
2. Write the tests, including a <?stf?> processing instruction in each.

   One practice is to use a "tests" directory containing a "go" subdirectory for tests that are expected to produce no Schematron assert or report messages and a "nogo" subdirectory for tests that are expected to have errors, but you can organise them any way you like.
3. Use Ant to run ${schematron} on files in ${test.dir}.

   You can run the test.schematron target from build.xml directly:

   ```
   ant -f /path/to/stf/build.xml test.schematron
   ```

   or you can import the stf "build.xml" into your local "build.xml":

   ```
   <property name="stf.dir" location="/path/to/stf" />
   <import file="${stf.dir}/build.xml" />
   ```

   or you can import the stf "build.xml" and include the <test.schematron/> macro in a target in your local "build.xml":

   ```
   <target name="test">
     <test.schematron schematron="tests.sch" />
     <xspec xspec.xml="tests.xspec" />
   </target>
   ```

Example Output

The stf output for running the Schematron on the two deliberately mismatched test documents shown above is shown below. Note that running the Schematron on the first file above produces a report for ERROR_BAR, but since that report is expected, stf does not report it as an error.

```
<errors>
  <result>
    <file>file:foo-1.xml</file>
    <error>Should be 1 reports or asserts for ERROR_QUX.
Found 0.</error>
    <error>Unexpected: ERROR_FOO:1</error>
  </result>
  <result>
    <file>file:foo-2.xml</file>
    <error>Should be no reports or asserts.
Unexpected: ERROR_FOO:1</error>
  </result>
</errors>
```

When the Schematron, the test documents, and their expected Schematron results are aligned, the stf output is:

```
<errors/>
```

MENTEA