Schematron Tutorial

Declarative Amsterdam 2022

Erik Siegel - Xatapult
2022-09-28

0 Table of Contents

1 Introduction and instructions

This tutorial is about Schematron. Schematron is an XML validation language for *business rules*. It can validate things far beyond the capabilities of DTD, XML Schema or RELAX NG.

High-level characteristics

- Schematron is a formal schema language in which you can express rules for XML documents.
- There are two types of rules:
    - Assertions: when the condition for an assertion fails, an error message is issued.
    - Reports: when the condition for a report holds, a report message is issued.
- In Schematron you define all the error and report messages in your own words.
- Schematron is expressed in XML: a Schematron schema is an XML document.
- Schematron allows you to specify the underlying language for its expressions. In practice, XPath is the only language supported.
- Schematron can, by design, incorporate constructs from other programming languages. However, most public implementations support XSLT only.

Why Schematron?

There are a number of reasons why Schematron is a very useful tool in the XML toolbox. Here are the most important ones:

- Schematron is a relatively simple but powerful validation language. Basic Schematron already has a wide field of application and is relatively easy to master.
- Schematron can go way beyond the validations of the “classic” validation languages like DTD, W3C XML Schema, and RELAX NG. It allows you to do extensive checks on XML structures and data that are not possible in other languages. Anything you can express as an XPath test can be used for validation purposes. More experienced users can take advantage of XSLT features such as keys and functions.
- In Schematron you define all the error or report messages yourself. For other validation languages you’re at the mercy of the validation processor’s implementer, and this often results in technically correct but, for users, obscure messages. In Schematron this is completely under your control. Messages can be enriched with computed text from or about the validated document by using XPath expressions.
- Since the messages are under your control, Schematron is often used to partially take over validations normally done by the other validation languages. Messages can be tailored to the user’s knowledge level or context.

Instructions for the exercises

- The easiest way to follow along with the exercises is by using oXygen. Please open the oXygen *project* `exercises/da-2022-schematron-exercises.xpr`. This project already associates the documents to validate with the appropriate Schematron schema for you.
- All exercises are in subfolders of the `exercises` folder called `exercise-xx-yy`
- Every exercise contains a PDF with instructions called `instructions.pdf`
- For most exercises there is a solution present in the `solution` subfolder.

1.1 Exercise 01-01 - Pre-flight check

Subfolder: `exercise-01-01`

We're going to validate the input document `input.xml`:

```
<inventory-list depcode="IMP">
  <article code="IMP0001">
    <name>Bolts</name>
    <description>Nuts to secure things with</description>
  </article>
  <article code="IMP0002">
    <name>Nuts</name>
    <description>Bolts to turn on the nuts</description>
  </article>
  <article code="EXP0234">
    <name>Bananas</name>
    <description>Delicious ripe bananas</description>
  </article>
</inventory-list>
```

The rule is that all codes must start with the value of `/*/@depcode`. The third (and last) article in the list breaks this rule.

A Schematron schema for checking this is in `schema.sch`:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="article">
      <assert test="starts-with(@code, /inventory-list/
@depcode)">The article code must start with the right prefix</assert>
    </rule>
  </pattern>
</schema>
```

Please validate `input.xml` with `schema.sch` and check the validation message.

2 Schematron fundamentals

2.1 Exercise 02-01

Subfolder: `exercise-02-01`
Exercise solution explanation: “Exercise 02-01” on page 6

We're going to validate the input document `input.xml`:

```
<DATA>
  <ARTICLE>
    <NAME>BOOK</NAME>
    <ID>ABC12345</ID>
  </ARTICLE>
  <ARTICLE>
    <NAME>TOY</NAME>
    <ID>XYZ123456</ID>
  </ARTICLE>
</DATA>
```

Rules are:

- Please *assert* that every identifier (in `<ID>` elements) is exctaly 9 characters long
- Please *report* identifiers (in `<ID>` elements) that start with the character `X` as being special

Use the template document `template.sch` as a starting point. In oXygen, the input document `input.xml` automatically uses `template.sch` for validation.

2.2 Exercise 02-02

Subfolder: `exercise-02-02`
Exercise solution explanation: “Exercise 02-02” on page 6

We're going to validate the input document `input.xml`:

```
<data>
  <book code="ABCD" pagecount="213"/>
  <magazine code="EFGH" articlecount="6"/>
  <!-- Invalid entries: -->
  <book code="ABCDX"/>
  <magazine code="EFGHX"/>
</data>
```

Rules are:

- A book must have a `pagecount` attribute
- A magazine must have an `articlecount` attribute
- All elements must have a code attribute of exactly 4 characters long

The Schematron schema in `template.sch` is incorrect:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="/*/book">
      <assert test="exists(@pagecount)">A book must have a pagecount attribute</assert>
    </rule>
    <rule context="/*/magazine">
      <assert test="exists(@articlecount)">A magazine must have an articlecount attribute</assert>
    </rule>
    <rule context="/*/*">
      <assert test="string-length(@code) eq 4">A code must be 4 characters long</assert>
    </rule>
  </pattern>
</schema>
```

The rule for *all* elements is not applied. Why?

Please verify this by validating `input.xml` against this schema (in oXygen this happens automatically when you open `input.xml`).

Improve the schema in `template.sch` so the rules for all elements are applied too.

2.3 Exercise 02-03

Subfolder: `exercise-02-03`
Exercise solution explanation: “Exercise 02-03” on page 6

This exercise builds on the previous one. We're going to validate the input document `input.xml` again:

```
<data>
  <book code="ABCD" pagecount="213"/>
  <magazine code="EFGH" articlecount="6"/>
  <!-- Invalid entries: -->
  <book code="ABCDX"/>
  <magazine code="EFGHX"/>
</data>
```

Rules are:

- A book must have a `pagecount` attribute
- A magazine must have an `articlecount` attribute
- All elements must have a code attribute of exactly 4 characters long

There's a basic Schematron schema in `template.sch`:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="/*/book">
      <assert test="exists(@pagecount)">A book must have a pagecount attribute</assert>
    </rule>
    <rule context="/*/magazine">
      <assert test="exists(@articlecount)">A magazine must have an articlecount attribute</assert>
    </rule>
  </pattern>
  <pattern>
    <rule context="/*/*">
      <assert test="string-length(@code) eq 4">A code must be 4 characters long</assert>
    </rule>
  </pattern>
</schema>
```

Please enhance this schema so all rules report the value of the `code` attribute somehow.

Additional changes

Please enhance the schema even further and assert that:

- Books have less than 200 pages
- Magazines have less than 6 articles

The messages for these rules must report both the book/magainze code and the incorrect number of pages/ articles.

3 Explanations

3.1 Schematron fundamentals

3.1.1 Exercise 02-01

One of the possible solutions for this exercise is in `solution/solution.sch`:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="ID">
      <assert test="string-length(.) eq 9">An ID must be 9 characters long!</assert>
      <report test="starts-with(., 'X')">Special identifier found!</report>
    </rule>
  </pattern>
</schema>
```

- The Schematron schema contains a single pattern with a single rule.
- This rule matches on every `<ID>` element.
- The assert tests whether the string length of this element is 9 characters long. If *not* it issues a message.
- The report tests whether the ID starts with the character X. if *so* it issues a message.

3.1.2 Exercise 02-02

One of the possible solutions for this exercise is in `solution/solution.sch`:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="/*/book">
      <assert test="exists(@pagecount)">A book must have a pagecount attribute</assert>
    </rule>
    <rule context="/*/magazine">
      <assert test="exists(@articlecount)">A magazine must have an articlecount attribute</assert>
    </rule>
  </pattern>
  <pattern>
    <rule context="/*/*">
      <assert test="string-length(@code) eq 4">A code must be 4 characters long</assert>
    </rule>
  </pattern>
</schema>
```

Every pattern is checked against every node in the document. So now both the specific rules for books and magazines (first `<pattern>` element) *and* the rules for all elements (second `<pattern>` element) will be applied.

3.1.3 Exercise 02-03

One of the possible solutions for this exercise is in `solution/solution.sch`:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="/*/book">
      <assert test="exists(@pagecount)">The book with code <value-of select="@code"/
> must have a pagecount attribute</assert>
    </rule>
    <rule context="/*/magazine">
      <assert test="exists(@articlecount)">The magazine with code <value-of select="@code"/
> must have an articlecount attribute</assert>
    </rule>
  </pattern>
  <pattern>
    <rule context="/*/*">
      <assert test="string-length(@code) eq 4">The code <value-of select="@code"/
> is invalide, it must be 4 characters long</assert>
    </rule>
  </pattern>
</schema>
```

It uses the `<value-of select="@code"/>` element to insert the value of the `code` attribute in the messages.

Additional changes

One of the possible solutions for the additional changes requested is in `solution/solution-extended.sch`:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <pattern>
    <rule context="/*/book">
      <assert test="exists(@pagecount)">The book with code <value-of select="@code"/
> must have a pagecount attribute</assert>
    </rule>
    <rule context="/*/magazine">
      <assert test="exists(@articlecount)">The magazine with code <value-of select="@code"/
> must have an articlecount attribute</assert>
    </rule>
  </pattern>
  <pattern>
    <rule context="/*/book/@pagecount">
      <assert test="xs:integer(.) lt 200">The pagecount of <value-of select="."/
> for book code <value-of select="../@code"/> must be less than 200</assert>
    </rule>
    <rule context="/*/magazine/@articlecount">
      <assert test="xs:integer(.) lt 6">The article count of <value-of select="."/
> for magazine code <value-of select="../@code"/> must be less than 6</assert>
    </rule>
  </pattern>
  <pattern>
    <rule context="/*/*">
      <assert test="string-length(@code) eq 4">The code <value-of select="@code"/
> is invalide, it must be 4 characters long</assert>
    </rule>
  </pattern>
</schema>
```

Please notice that the checks for the page and article count are in a pattern on their own. The rules fire on the *attribute* (so not on the element). This makes sure these rules don't apply to (invalid) books and magazines that have no `pagecount`/`articlecount` attribute.

Also notice the use of the `xs:integer()` function to explicitly convert the value of the attribute to an integer. A numerical comparison will not work without this!