

# **Schematron Tutorial**

**Declarative Amsterdam 2022**



## 0 Table of Contents

<b>1 Introduction and instructions</b>	2
1.1 Exercise 01-01 - Pre-flight check	2
<b>2 Schematron fundamentals</b>	4
2.1 Exercise 02-01 - Using patterns, rules, asserts and reports	4
2.2 Exercise 02-02 - Rules processing	4
2.3 Exercise 02-03 - Enhancing messages	5
2.4 Exercise 02-04 - Using variables	5
2.5 Exercise 02-05 - Using namespaces	6
<b>3 Abstract patterns</b>	7
<b>4 Query Language Binding</b>	8
4.1 Exercise 04-01 - Use an XSLT function	8
<b>5 Explanations</b>	9
5.1 Schematron fundamentals	9
5.1.1 Exercise 02-01 - Using patterns, rules, asserts and reports	9
5.1.2 Exercise 02-02 - Rules processing	9
5.1.3 Exercise 02-03 - Enhancing messages	9
5.1.4 Exercise 02-04 - Using variables	10
5.1.5 Exercise 02-05 - Using namespaces	10
5.2 Query Language Binding	11
5.2.1 Exercise 04-01 - Use an XSLT function	11

# 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 deptime="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 `/*/@deptime`. 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/
@deptime)">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 - Using patterns, rules, asserts and reports

Subfolder: exercise-02-01

Exercise solution explanation: "Exercise 02-01 - Using patterns, rules, asserts and reports" on page 9

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 exactly 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 - Rules processing

Subfolder: exercise-02-02

Exercise solution explanation: "Exercise 02-02 - Rules processing" on page 9

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 - Enhancing messages

Subfolder: exercise-02-03

Exercise solution explanation: “Exercise 02-03 - Enhancing messages” on page 9

This exercise builds on the previous exercise 02-02. 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/magazine code and the incorrect number of pages/articles.

Make sure these additional checks don't fire on books/magazines that are invalid and have no `pagecount`/`articlecount` attribute.

## 2.4 Exercise 02-04 - Using variables

Subfolder: exercise-02-04

Exercise solution explanation: “Exercise 02-04 - Using variables” on page 10

This exercise builds on the pre-flight check exercise 01-01. We're validating the input document `input.xml`:

```
<inventory-list xmlns="http://www.xtpxlib.nl/ns/xcourse" deptime="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 for this document is that all codes must start with the department code, which is in the `deptime` attribute on the root element.

The basic Schematron schema for this is in `template.sch`:

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

Enhance this schema so that:

- The department code is stored in a global variable and used in the assert's expression and message
- Add a variable to the rule that stores the value of the article's `code` attribute and use this in the assert's expression and message
- Also add the name of the article (in the `<name>` element) to the message

## 2.5 Exercise 02-05 - Using namespaces

Subfolder: `exercise-02-05`

Exercise solution explanation: "Exercise 02-05 - Using namespaces" on page 10

In this exercise we're going to validate the DocBook input document `input.xml`:

```
<article xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>An example document</title>
  <sect1>
    <title>The first section about the number 42 (which has a very, very, very long title indeed)</title>
    <para>A paragraph of text.</para>
    <para>A <emphasis>second</emphasis> paragraph of text!</para>
  </sect1>
</article>
```

The rules for this document are:

- A section on the first level (`<sect1>` elements) must have at least 3 paragraphs (`<para>` elements)
- The title of any section must not be longer than 30 characters

A template Schematron schema for this is in `template.sch`. Finish this schema so the rules above are validated.

As an extra, please use variables for the magic values, 3 and 30, in the validation rules.



### **3 Abstract patterns**

## 4 Query Language Binding

### 4.1 Exercise 04-01 - Use an XSLT function

Subfolder: exercise-04-01

Exercise solution explanation: "Exercise 04-01 - Use an XSLT function" on page 11

This exercise builds upon the first 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>
```

Please change the schema:

- The test for the `code` attribute (`starts-with(@code, /inventory-list/@depcode)`) must be moved to a separate (boolean) XSLT function.
- Use this function in the assert's test.

Remark: to save you from having to type the XSLT namespace correctly, it is already declared on the template's root element.

## 5 Explanations

### 5.1 Schematron fundamentals

#### 5.1.1 Exercise 02-01 - Using patterns, rules, asserts and reports

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.

#### 5.1.2 Exercise 02-02 - Rules processing

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.

#### 5.1.3 Exercise 02-03 - Enhancing messages

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 invalid, 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 invalid, 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/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!

## 5.1.4 Exercise 02-04 - Using variables

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

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

- The first variable, `department-code` stores the global value of the `depcode` attribute on the root element.
- The rule creates a variable `article-code` with the value of article's the `code` attribute.
- Both variables are used in both the assert's expression and resulting message.

## 5.1.5 Exercise 02-05 - Using namespaces

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

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <ns prefix="db" uri="http://docbook.org/ns/docbook"/>
  <let name="title-max-length" value="30"/>
  <let name="sect1-min-para-count" value="3"/>
  <pattern>
    <rule context="db:sect1">
      <assert test="count(db:para) ge $sect1-min-para-count">The section titled "<value-of select="db:title"/>" must contain at least
        <value-of select="$sect1-min-para-count"/> paragraphs of text</assert>
    </rule>
  </pattern>
  <pattern>
    <rule context="db:title">
      <let name="title-length" value="string-length(.)"/>
      <assert test="$title-length le $title-max-length">The title "<value-of select="."/>" is <value-of select="$title-length"/> characters
        long, which is longer than the maximum allowed <value-of select="$title-max-length"/> characters</assert>
    </rule>
  </pattern>
</schema>
```

- The `<ns>` element defines the DocBook namespace and assigns it the prefix `db`
- Both magic values, 3 and 30, are stored in variables
- The first pattern checks the paragraph count of DocBook `<sect1>` elements
- The second pattern checks the length of any DocBook `<title>` element

Please notice that the input document uses a default namespace declaration (`xmlns="http://docbook.org/ns/docbook"`) and the Schematron schema a namespace prefix (`db`). As long as the namespace URI is the same in both cases (`http://docbook.org/ns/docbook`) this does not matter.

## 5.2 Query Language Binding

### 5.2.1 Exercise 04-01 - Use an XSLT function

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

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt3">
  <ns prefix="f" uri="#functions"/>
  <let name="department-code" value="*/@depcode"/>
  <xsl:function xmlns:xsl="http://www.w3.org/1999/XSL/Transform" name="f:check-code" as="xs:boolean">
    <xsl:param name="code" as="xs:string"/>
    <xsl:sequence select="starts-with($code, $department-code)"/>
  </xsl:function>
  <pattern>
    <rule context="article">
      <assert test="f:check-code(@code)">The article code must start with the right prefix</assert>
    </rule>
  </pattern>
</schema>
```

- Since we're going to use XSLT elements in our Schematron schema, we have to define the XSLT namespace on the root element (and bind it here to the prefix `xsl`)
- Define some namespace for the function (`#functions`) and assign it a prefix (`f`). As long as its URI doesn't clash with any of the other namespaces in use here, it doesn't matter what it is
- We get the department code upfront. This is necessary here because an XPath function has no context item and therefore has no access to the document being validated
- Define a very simple function (`f:check-code`) with one parameter
- And use this function in the assert's test expression