A hands-on introduction to Schematron

Directly express rules without creating a whole grammatical infrastructure

Skill Level: Intermediate

Uche Ogbuji (uche.ogbuji@fourthought.com)
Consultant and Co-Founder
Fourthought

02 Sep 2004

Meet Schematron, a schema language that allows you to directly express rules without creating a whole grammatical infrastructure. Schematron is useful whenever you wish to apply and check against rules for the contents of XML documents. Schematron is extraordinarily flexible in the variety of rules you can express, and it's even more expressive than other schema languages such as DTD, W3C XML Schema (WXS) and RELAX NG. In this tutorial, author Uche Ogbuji uses detailed examples to illustrate Schematron's use, and offers recipes for common schema needs.

Section 1. Tutorial introduction

Who should take this tutorial?

Most developers of XML-based applications want a way to be sure that all XML instances follow certain rules. For this, many people immediately turn to schema languages such as DTD, W3C XML Schema (WXS), and RELAX NG. The best solution might be to apply simple rules to XML instances. Schematron is a language that allows you to directly express rules without creating a whole grammatical infrastructure.

All developers of XML vocabularies and software that uses XML in any significant way should learn Schematron, and this tutorial is a great way to get started. Even if you already use schema languages such as RELAX NG and WXS, you may need to augment them with Schematron rules, which are more general and more flexible.

Prerequisites

This tutorial assumes knowledge of XML, XML Namespaces, and XPath. If you aren't familiar with these technologies, I recommend you first take these *developerWorks* tutorials:

- "Introduction to XML"
- "Get started with XPath"

It is helpful to have some knowledge of XSLT -- at least the basics of how to declare and apply templates. If you aren't familiar with XSLT, I recommend that you first take the tutorial "Create multi-purpose Web content with XSLT".

I highly recommend that you follow along with the examples (see Resources to download it). To do so you will need to use a Schematron tool of some sort. While developing these examples I used the Scimitar implementation of ISO Schematron. Other tools are available from the Schematron resource page (see Resources).

About the Schematron examples in this tutorial

In this tutorial you will see many examples of Schematron files, and XML documents that illustrate the sorts of patterns the schemas are looking for. All the files used in this tutorial are in the zip file, x-schematron-files.zip (see Resources). In this package, all files start with a prefix indicating the tutorial section that covers them and the order of examples within the section. For example, files from the first example in the third section are named starting with "eg3_1".

Files ending with ".sch" are Schematron schema files, and files ending in ".xml" are sample XML documents to be processed with the schema of the same prefix. Some, such as eg3_1_good1.xml, are valid against the corresponding schema; some, such as eg3_1_bad1.xml, are not valid. A few, such as eg4_1_1.xml, correspond to Schematron files that are not meant for validation at all, but rather for reporting.

I do take care to further list the example files in each panel featuring an example. If you follow along with the tutorial you should be able to locate and experiment with the examples easily enough.

Section 2. Schematron overview and example

The problems to be solved

Schematron is useful whenever you wish to apply and check against rules for the contents of XML documents. Schematron is extraordinarily flexible in the variety of rules you can express, and it's even more expressive than other schema languages such as DTD, W3C XML Schema (WXS), and RELAX NG.

This tutorial illustrates this power with a sample scenario of an organization that publishes technical documents that are submitted by diverse authors. The editors wish to test each submission against a set of rules to ensure that the documents meet editorial standards, and that the publishing tools properly process the documents.

This tutorial introduces Schematron by example, and provides recipes for common schema needs. Again, the emphasis is on tasks, so I highly recommend that you fire up a Schematron toolkit and follow along, hands-on.

The anatomy of Schematron

A Schematron schema is made up of elements in the `http://www.ascc.net/xml/schematron` namespace. The root element has a local name of `schema`. You can represent this in any of the many ways that are made available by XML namespaces. In this tutorial I shall make Schematron's namespace the default so that no prefixes are necessary.

The `schema` element should have a descriptive `title` element. At its heart, the schema has a number of `rule` elements.

Each rule contains a set of individual checks represented using `assert` or `report` elements. Rules are organized using `pattern` elements that contain collections of related `rule` elements. Rules and patterns can also have descriptive `title` elements.

This is just a high-level description. I shall elaborate much more on each element in the examples. Schematron has a few other elements that you can use, but the ones I've described here are the heart of the language, and by far the most common. Indeed, Schematron is amazingly simple -- and as you shall learn, it is also amazingly powerful.

The next panel, Sample Schematron schema, gives you a picture of what Schematron looks like. Again, I'll explore much more on the various details in rest of the tutorial.

Sample Schematron schema

Here's an example of a schema that checks on XHTML usage

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Special XHTML conventions</title>
  <ns uri="http://www.w3.org/1999/xhtml" prefix="html"/>
  <pattern name="Document head">
    <rule context="html:head">
      <assert test="html:title">Page does not have a title.</assert>
      <assert test="html:link/@rel = 'stylesheet'
                    and html:link/@type = 'text/css'
                    and html:link/@href = 'std.css'">
        Page does not use the standard stylesheet.
      </assert>
      <report test="html:style">
        Page uses in-line style rather than linking to the
        standard stylesheet.
      </report>
    </rule>
  </pattern>
  <pattern name="Document body">
    <rule context="html:body">
      <assert test="@class = 'std-body'">
        Page does not use the standard body class.
      </assert>
      <assert test="html:*[1]/self::html:div[@class = 'std-top']">
        Page does not start with the required page top component.
      </assert>
    </rule>
  </pattern>
</schema>
```

Walk-through of the example schema rules

In plain language, the rules expressed in the Sample Schematron schema are basically:

- The document head contains a `title` element.
- A `stylesheet` element loads the prescribed cascading stylesheet (CSS).
- The schema notes if a `style` element is present in the document head.
- The document body is declared in the `std-body` class.
- The document body starts with a special `div` in a class named `std-top`.

An XML document to be validated with the Schematron rules is a **candidate** instance or document.

For reference, have a look at the next panel, A sample valid document : It includes a candidate instance that follows all of these rules, and thus should result in no errors when checked against Sample Schematron schema.

A sample valid document

Here's an example of a schema that checks a few XHTML conventions:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <link rel="stylesheet" type="text/css" href="std.css"/>
    <title>Document head</title>
  </head>
  <body class="std-body">
    <div class="std-top">Top component</div>
  </body>
</html>
```

Section 3. Basics of rules, patterns, and assertions

What are patterns and rules?

A pattern is a collection of related rules. A Schematron processor operates by examining in document order each node in the candidate instance. For each element, it checks all the patterns, and executes rules that are appropriate for that element. It executes no more than one rule in each pattern.

Each rule has a `context` attribute that determines which elements will trigger that rule. It does this in the same way that templates match context nodes in XSLT -- in fact, the value of the `context` attribute is a standard XSLT **XPattern**. XPattern is a subset of XPath.

In most cases for Schematron, you will keep things simple and just use what looks like an XPath to match an element by qualified name (or `/` to match the root node). For example, the rule in the following snippet matches an XHTML `head` element.

```

  <pattern name="Document head">
    <rule context="html:head">

    </rule>
  </pattern>
```

The `html:head` context basically says "fire this rule when the Schematron processor gets to a `head` element in XHTML namespace."

Example schema for assertions

When a rule is fired, the Schematron processor checks for **assertions** and **reports** declared in the body of the rule. An assertion is an XPath expression that you expect to evaluate to `true` using the rule's context in a valid document. An `assert` element has a `test` attribute with the XPath expression; the body of the element is a brief message that expresses the condition that is expected to be true.

Now it's time to start working within the example usage scenario I outlined earlier.

This schema checks that candidate documents have root elements named `doc` (in no namespace).

Notice that the context is `/`, which matches the root node, allowing the rule to check conditions relating to the root element.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Document root">
    <rule context="/">
      <assert test="doc">Root element should be "doc".</assert>
    </rule>
  </pattern>
</schema>
```

Executing the sample schema

To be sure you get the benefit of the examples, please use a Schematron implementation to run the Example schema for assertions. It's in the download package as **eg3_1.sch**. **eg3_1_good1.xml** is a file with the simple contents `<doc/>`, while **eg3_1_bad1.xml** has the simple contents `<bogus/>` . The former should validate cleanly against Example schema for assertions, while the latter should cause a failure in the assertion, signaling you of the failure in some way.

For example, if you install the Scimitar implementation, you first prepare the schema by compiling it into a reusable validator program:

```
$ scimitar.py eg3_1.sch
```

This creates the validator program **eg3_1-stron.py** which you can run against the candidate documents. First the valid one:

```
$ python eg3_1-stron.py eg3_1_good1.xml
<?xml version="1.0" encoding="UTF-8"?>
Processing schema: Technical document schema
Processing pattern: Document root
```

Now the invalid candidate:

```
$ python eg3_1-stron.py eg3_1_bad1.xml
<?xml version="1.0" encoding="UTF-8"?>
Processing schema: Technical document schema
Processing pattern: Document root
Assertion failure:
Root element should be "doc".
```

The assertion failure message is in bold.

Validating the presence of elements

You can validate that certain elements are present. This schema checks that `doc` elements have both `prologue` and `section` elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Major elements">
    <rule context="doc">
      <assert test="prologue">
        The "doc" element must have a "prologue" child.
      </assert>
      <assert test="section">
        The "doc" element must have at least one "section" child.
      </assert>
    </rule>
  </pattern>
</schema>
```

It's worth pointing out that most of the examples in this section are quite simple, and designed to give you a basic grasp of common validation patterns. Most of these validation tasks could be just as easily (and sometimes less verbosely) expressed in other schema languages. In later sections, I shall touch on some examples that illustrate the unique strengths of Schematron.

This is **eg3_2.sch** in x-schematron-files.zip. Run it against:

- **eg3_2_good1.xml**, which has one of each required element
- **eg3_2_good2.xml**, which has an extra `section` element
- **eg3_2_bad1.xml**, which is missing the `prologue` element
- **eg3_2_bad2.xml**, which is missing the `section` element
- **eg3_2_bad3.xml**, which is missing both

Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating that elements are where expected

To validate that an element appears only in a certain place, use this schema to check that the only `doc` element is the root.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Extraneous docs">
    <rule context="doc">
      <assert test="not(ancestor::*)">
        The "doc" element is only allowed at the document root.
      </assert>
    </rule>
  </pattern>
</schema>
```

The key here is the XPath `not(ancestor::*)`, which means "the context node has no element ancestors."

The code listing above is **eg3_3.sch** in x-schematron-files.zip. Run it against **eg3_3_good1.xml**, which only uses a doc element in the correct place, and **eg3_3_bad1.xml**, which has an improper extra doc element. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating relative positioning of elements

You can validate that a certain element comes before or after another. This schema checks that the `prologue` comes before any `section` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Extraneous docs">
    <rule context="prologue">
      <assert test="not(preceding-sibling::section)">
        No "section" may occur before the "prologue".
      </assert>
    </rule>
  </pattern>
</schema>
```

The XPath `preceding-sibling` and `following-sibling` axes are key to positioning assertions. As a reminder, they cover all nodes with the same parent that come before or after the context node. Relative positioning is an example of a constraint that is much more easily expressed in Schematron than in other languages; those languages usually allow you to define an explicit element sequence, but don't let you check more general cases.

The code listing above is **eg3_4.sch** in the x-schematron-files.zip. Run it against **eg3_4_good1.xml**, which has a prologue followed by two sections, and **eg3_4_bad1.xml**, which has a prologue between two sections. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating a sequence of elements

You can validate that elements occur in a specified, immediate sequence. This schema checks that a `title` element is immediately followed by a `subtitle` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Title with subtitle">
    <rule context="title">
      <assert test="following-sibling::*[1]/self::subtitle">
        A "title" must be immediately followed by a "subtitle".
      </assert>
    </rule>
  </pattern>
</schema>
```

The XPath `following-sibling::*[1]/self::subtitle` may seen a bit daunting at first glance, but if you break it down it should make perfect sense. The first step, `following-sibling::*[1]`, selects the element immediately following the context (`title`). The next step, `self::subtitle`, ensures that this element is a `subtitle`. In many uses of XPath, not just Schematron, the `self` axis is the key to a lot of power, and you should make a habit of putting it to work.

The code listing above is **eg3_5.sch** in x-schematron-files.zip. Run it against:

- **eg3_5_good1.xml**, which has a proper title and subtitle element
- **eg3_5_bad1.xml**, which is missing a subtitle completely
- **eg3_5_bad2.xml**, which has both elements, but with an extraneous element between them

Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating for a certain number of elements

You can validate whether there is a specific number of a particular element present. To make it easier for readers to find the article in relevant contexts, the journal editors want to be sure that articles have at least three keywords in the prologue. This schema enforces a minimum of three `keyword` children of the `prologue` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Minimum keywords">
    <rule context="prologue">
      <assert test="count(keyword) > 2">
        At least three keywords are required.
      </assert>
    </rule>
  </pattern>
</schema>
```

The code listing above is **eg3_6.sch** in x-schematron-files.zip. Run it against **eg3_6_good1.xml**, which has three keywords, and **eg3_6_bad1.xml**, which has only two. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating presence and value of attributes

You can validate that an attribute appears, or that it has a certain value. This schema checks that an `author` element (child of `prologue`) has an `e-mail` attribute and a `member` attribute. The latter indicates whether or not an author is a member of the technical association, and this schema checks that its value is "yes" or "no."

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Author attributes">
    <rule context="author">
      <assert test="@e-mail">
        author must have e-mail attribute.
      </assert>
      <assert test="@member = 'yes' or @member = 'no'">
        author must have member attribute with 'yes' or 'no' value.
      </assert>
    </rule>
  </pattern>
</schema>
```

This code listing is **eg3_7.sch** in x-schematron-files.zip. Run it against:

- **eg3_7_good1.xml**, which has the required attributes
- **eg3_7_bad1.xml**, which is missing `e-mail`
- **eg3_7_bad2.xml**, which is missing `member`
- **eg3_7_bad3.xml**, which has a bad value for `member`

Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Simple validation of element content

To validate that an element has a certain value, use this schema to check that the `title` element is not empty.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Useful title">
    <rule context="title">
      <assert test="text()">
        title may not be empty.
      </assert>
    </rule>
  </pattern>
</schema>
```

The XPath `text()` checks for any child text. If `title` could contain child elements (such as for emphasis) and you want to check that `title` contains text somewhere in its content, change the test to `.//text()` so you use the descendant-or-self axis.

The code listing above is **eg3_8.sch** in x-schematron-files.zip. Run it against **eg3_8_good1.xml**, which has a good title, and **eg3_8_bad1.xml**, which has an empty title. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating exclusivity of elements

You can validate that no unwanted elements are present. This schema checks that `author` elements only have `name`, `bio`, and `affiliation` elements as children.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Author elements">
    <rule context="author">
      <assert test="count(name|bio|affiliation) = count(*)">
        Only "name", "bio" and "affiliation" elements are allowed
        as children of "author".
      </assert>
    </rule>
  </pattern>
</schema>
```

The XPath `count(name|bio|affiliation) = count(*)` compares the count of all elements with the count of just those expected. If these differ, then an extraneous element is present.

This code listing is **eg3_9.sch** in x-schematron-files.zip. Run it against **eg3_9_good1.xml**, which has only the required elements, and **eg3_9_bad1.xml**, which has an unwanted element. Keep in mind that these are just examples and don't necessarily represent solid XML design. For example, you might want to use better structure for names in XML documents. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Section 4. Reports and communications control

More than just validation

One of the strengths of Schematron is that it is not just a validation language. In general, it is a framework for doing all of the useful things that can be defined using the sorts of rules I covered in the previous section, Basics of rules, patterns, and assertions . This includes validation, but also includes reporting. In fact, I tend to think of Schematron more as an XML reporting language, with one possible type of report being validation errors.

You have seen how the `assert` instruction allows you to express validation constraints. `report` is a very similar Schematron instruction that is intended to allow you to trigger more general reporting tasks. I present an example of `report` in the next panel, Reports.

Several aspects of the framework nature of Schematron are worth bearing in mind. The first is that Schematron doesn't govern the means of presenting output to users. I have discussed Scimitar, an implementation that prints output to the command line. A Schematron implementation could also be an interactive forms application. Output from Schematron reporting can include XML tags, as you will see later on in this section.

Another thing to bear in mind is that XPath 1.0 isn't the only query language you can use in assertion tests. ISO Schematron allows you to add in XPath extension functions such as those in EXSLT (see Resources), and you can also use XPath 2.0, XQuery, or even non-XML, if your Schematron implementation supports this. However, I do not know of any implementation that's based on anything other than XPath 1.0, and I do not cover alternative query languages in this tutorial.

Reports

Use the `report` instruction to communicate information about the XML instance, apart from what would generally be considered a validation error. `report` is processed very similarly to `assert` except that the report message is triggered when the boolean value of the `test` attribute is true, rather than false. This schema issues a report if the author's e-mail is in the US military network domain.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Military authors">
    <rule context="author">
      <!-- This test should really be refined so it
           checks that '.mil' is in the last position. -->
      <report test="contains(@e-mail, '.mil')">
        Author appears to be military personnel.
      </report>
    </rule>
  </pattern>
</schema>
```

This is **eg4_1.sch** in x-schematron-files.zip. Run it against **eg4_1_1.xml**, which has an author e-mail in the .mil domain, and **eg4_1_2.xml**, which does not. Run these using your Schematron implementation of choice, and experiment with the schema and candidate documents.

Using the context node's name in messages

Validation rules often apply to more than one element. In such cases, it is convenient to be able to dynamically determine the element's name when the message is being processed. As an example, if the rule is that "all XHTML elements must have a `class` attribute," it wouldn't be helpful to get a message saying: "some element in the document is missing a `class` attribute." It would be more useful to get a message such as: "a `blockquote` element is missing a `class` attribute." Schematron provides a `name` instruction for this purpose, which you can use within the body of `assert` and `report`. The following schema sends a report if it comes across any element with a `link` attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Report links">
    <rule context="*">
      <report test="@link">
        <name/> element has a link.
      </report>
    </rule>
  </pattern>
</schema>
```

This is **eg4_2.sch** in x-schematron-files.zip. Run it against **eg4_2_1.xml**, which has a link. Run it using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Using a specified node's name in messages

You may wish to dynamically insert a node's name into the output, but not necessarily that of the context node. You can do so by using the `path` attribute with `name` . The following is a variation on Validating a sequence of elements which gives a more precise assertion message.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Title with subtitle">
    <rule context="title">
      <assert test="following-sibling::*[1]/self::subtitle">
        A <name/> must be immediately followed by a "subtitle",
        not <name path="following-sibling::*[1]"/>.
      </assert>
    </rule>
  </pattern>
</schema>
```

The code listing above is **eg4_3.sch** in x-schematron-files.zip. Run it against:

- **eg4_3_good1.xml**, which has a proper title and subtitle element
- **eg4_3_bad1.xml**, which is missing a subtitle completely
- **eg4_3_bad2.xml**, which has both elements but includes an extraneous element between them, whose name will be reflected in the message

Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Using an arbitrary XPath in messages

Sometimes you need even more expressive power in messages than `name` offers. Schematron's `value-of` instruction is much like the XSLT instruction of the same name. You can place any XPath expression in its `select` attribute, and the expression will then be evaluated in the context of the current rule and converted to string value, which is inserted into the message. You can use `value-of` within the body of `assert` and `report`. The following schema sends a report if it comes across any element with a `link` attribute, specifying the link target.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Report links">
    <rule context="*">
      <report test="@link">
        <name/> element has a link to <value-of select="@link"/>.
      </report>
    </rule>
  </pattern>
</schema>
```

This is **eg4_4.sch** in x-schematron-files.zip. Run it against **eg4_4_1.xml**, which has a link. Run this using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Diagnostic messages

Schematron assertions are designed to be general statements of expectation, rather than instructions for addressing validation problems. Schematron lets you augment these brief messages by allowing you to associate diagnostic messages with `assert` and `report` instructions. You can do this by defining top-level `diagnostic` elements with the desired diagnostic message. Each of these elements must have an `id` attribute. `assert` and `report` can have a `diagnostics` attribute with a white space delimited list of diagnostic message IDs. If the assertion or report message is triggered, the Schematron processor will also output any specified diagnostic message.

`diagnostic` instructions can have `name` and `value-of` , as covered in Using the context node's name in messages , Using a specified node's name in messages , and Using an arbitrary XPath in messages.

The following example schema provides useful messages to guide the user in addressing validation errors.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
```

```
  <title>Technical document schema</title>
  <pattern name="Major elements">
    <rule context="doc">
      <assert test="prologue" diagnostics="doc-struct">
        <name/> element must have a prologue.
      </assert>
      <assert test="section" diagnostics="doc-struct sect">
        <name/> element must have at least one section.
      </assert>
    </rule>
  </pattern>
  <diagnostics>
    <diagnostic id="doc-struct">
       A document (the <name/> element) must have a prologue and one or
       more sections.  Please correct your submission by adding the
       required elements, then re-submit.  For your records, the
       submission ID is <value-of select="@id"/>.
    </diagnostic>
    <diagnostic id="sect">
       Sections are sometimes omitted because authors think they can
       just place the document contents directly after the prologue.
       You may be able to correct this error by just wrapping your
       existing content in a section element.
    </diagnostic>
  </diagnostics>
</schema>
```

This listing is **eg4_5.sch** in x-schematron-files.zip. Run this against:

- **eg4_5_good1.xml**, which has one of each required element
- **eg4_5_good2.xml**, which has an extra `section` element
- **eg4_5_bad1.xml**, which is missing the `prologue` element
- **eg4_5_bad2.xml**, which is missing the `section` element
- **eg4_5_bad3.xml**, which is missing both

Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Section 5. Intermediate Schematron features

Querying namespaces

Schematron provides full support for XML namespaces. To declare a namespace for use in rules, add an `ns` instruction as a child of the `schema`. Give it a `prefix` with the namespace prefix to be used within the schema, and a `uri` with the namespace name (a URI). As I have mentioned, the prefix you declare in the schema is *only used to resolve namespaces within expressions in the schema, not in the candidate document*. For example, if the candidate is an XHTML document, it would not use a prefix for XHTML elements, but you must declare a prefix such as "html" in order to match XHTML elements in your schema. XPath and XPattern (used in rule contexts) require that you use prefixes for all namespace-aware node expressions. The following schema validates that an XHTML document has a title:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Special XHTML conventions</title>
  <ns uri="http://www.w3.org/1999/xhtml" prefix="html"/>
  <pattern name="Document head">
    <rule context="html:head">
      <assert test="html:title">
          Page does not have a title.
      </assert>
    </rule>
  </pattern>
</schema>
```

This code listing is **eg5_1.sch** in x-schematron-files.zip. Run it against **eg5_1_good1.xml**, which has a title, and **eg5_1_bad1.xml**, which does not. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Using namespaces in output

As I mentioned, you can basically think of Schematron as a reporting framework. Toward this end, Schematron also supports using elements and attributes in output. Any element that appears in an output message but not in the Schematron namespace gets copied to the output as is. This example is the same as that in Validating the presence of elements, except with XHTML elements used for output.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron"
        xmlns:html="http://www.w3.org/1999/xhtml">
  <title>Technical document schema</title>
  <pattern name="Major elements">
    <rule context="doc">
      <assert test="section">
        <html:p>
          <name/> must have at least one <html:code>section</html:code>
          child.
        </html:p>
      </assert>
    </rule>
  </pattern>
</schema>
```

Notice the new namespace declaration on the root element. Such namespace declarations are only used for output elements. As discussed in Querying namespaces, if you want to use namespaces in queries, you must use the `ns` instruction for declaration. This means that if you are querying XHTML elements as well as using XHTML in output, you have to declare the namespace twice, using both mechanisms.

This listing is **eg5_2.sch** in x-schematron-files.zip. Run it against **eg5_2_good1.xml**, which has the required element, and **eg5_2_bad1.xml**, which doesn't. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Keys: An introduction

DTDs allow you to associate one element with another by using attributes of type `ID` and `IDREF`, which make up a mechanism so limited it hasn't received much use in industry practice. Other schema languages provide somewhat better ways to tie elements and attributes together, but Schematron provides unique power and flexibility by borrowing XSLT's key facility.

You can create a Schematron key by using a `key` instruction within the `schema`. It includes:

- A `use` attribute
- An XPath that gives the value for the key item
- A `match` attribute, which determines what nodes are covered
- A `name` attribute -- a simple string that gives the name of the key

The Schematron processor gathers all the nodes in the candidate document that match the XPattern given in `match`, and creates a look-up table with the given name. The key of each row in the look-up table (the look-up string) is the result of evaluating `use` against the matched nodes, and the value is a list of nodes with same look-up string.

You can access any keys you have defined in XPath expressions using the `key` function, which takes two parameters: the name of the key and the look-up string. The result is a node set with all nodes from the table corresponding to the look-up string. The example in the next panel, Keys: An example, helps illustrate this.

Keys: An example

You can use keys to check the reference of one value in a document against other values. In this example of keys, I refer back to the technical submissions scenario. A `main-contact` element is allowed in the prologue, with the restriction that its `e-mail` attribute must match the same attribute in one of the authors.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <key name="author-e-mails" match="author" use="@e-mail"/>
  <pattern name="Main contact">
    <rule context="main-contact">
      <assert test="key('author-e-mails', @e-mail)">
        "e-mail" attribute must match the e-mail of one of the authors.
      </assert>
    </rule>
  </pattern>
</schema>
```

The key definition maps each author's e-mail address to the author node. The key is invoked in the assertion check by looking up the e-mail used for the `main-contact`; if the look-up fails, the result is an empty node set, which is converted to boolean as false, and causes the assertion to fail.

This listing is **eg5_3.sch** in x-schematron-files.zip. Run it against **eg5_3_good1.xml**, which has a valid main contact, and **eg5_3_bad1.xml**, whose main contact does not match any author. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Validating based on conditions in the document

Very often you'll want to validate one part of a document based on what occurs in another part. This is something called a co-occurrence constraint. WXS and DTD cannot handle such validation at all, and RELAX NG can handle only limited examples, but Schematron provides extraordinary power for such validation tasks. This schema checks that content by each author includes at least three sections, with the goal of encouraging longer submissions and discouraging people from padding the author list.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <pattern name="Section minimum">
    <rule context="doc">
      <assert test="count(section) >= 3*count(prologue/author)">
        There must be at least three sections for each author.
      </assert>
    </rule>
  </pattern>
</schema>
```

When using Schematron, don't think in terms of other schema languages, or you probably won't take advantage of all its power. Just think of what rules you'd like to express about the candidate document, and chances are you'll be able to find a way to express it using XPath, and thus in Schematron.

The code listing above is **eg5_4.sch** in x-schematron-files.zip. Run it against **eg5_4_good1.xml**, which meets the section count minimum, and **eg5_4_bad1.xml**, which does not. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Phases

If I were to combine all the rules in this tutorial into one Schematron schema, it would be a large one. Schematron allows for modularity of schemata by allowing patterns to be organized into phases. A phase is a simple collection of patterns that are executed together. Some Schematron implementations allow you to select a particular phase to process. The following large sample schema incorporates several of the example rules from this tutorial, and organizes them into phases.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Technical document schema</title>
  <key name="author-e-mails" match="author" use="@e-mail"/>
  <phase id="quick-check"> <!-- "minimal sanity check" -->
    <active pattern="rightdoc" />
  </phase>
  <phase id="full-check">
    <active pattern="rightdoc" />
    <active pattern="extradoc" />
    <active pattern="majelements" />
  </phase>
  <phase id="process-links">
    <active pattern="report-link" />
  </phase>
  <pattern id="rightdoc" name="Document root">
    <rule context="/">
      <assert test="doc">Root element must be "doc".</assert>
    </rule>
  </pattern>
  <pattern id="extradoc" name="Extraneous docs">
    <rule context="doc">
      <assert test="not(ancestor::*)">
        The "doc" element is only allowed at the document root.
      </assert>
    </rule>
  </pattern>
  <pattern id="majelements" name="Major elements">
    <rule context="doc">
      <assert test="prologue">
        <name/> must have a "prologue" child.
      </assert>
      <assert test="section">
        <name/> must have at least one "section" child.
      </assert>
    </rule>
  </pattern>
  <pattern id="report-link" name="Report links">
    <rule context="*">
      <report test="@link">
        <name/> element has a link to <value-of select="@link"/>.
      </report>
    </rule>
  </pattern>
</schema>
```

The code is **eg5_5.sch** in x-schematron-files.zip. To trigger various validity messages and reports, run it against the various files **eg5_5_good*x*.xml** and **eg5_5_bad*x*.xml**, where x indicates a number that specifies a particular file. The files include documents that should trigger various validity messages and reports. Run these using your Schematron implementation of choice, and experiment with the schema and the candidate documents.

Section 6. Wrap up

Summary

In this tutorial you learned that Schematron is an expressive and flexible reporting framework that you can use as a powerful validation language. You learned how Schematron arranges validation assertions and reports into rules, which make up patterns that can be organized into phases. You learned how to execute the most common validation tasks using Schematron, as well as how to express some constraints that you may not have expected to express in a schema language.

Resources

Learn

- Learn the fundamentals of XPath with the tutorial "Get started with XPath" (*developerWorks*, May 2004), by Bertrand Portier.
- Get familiar with XSLT -- take the *developerWorks* tutorial "Create multi-purpose Web content with XSLT" by Nicholas Chase (March 2003). As another option, "Python and XML development using 4Suite, Part 2: 4XPath and 4XSLT" (October 2001) includes an introduction to XPath and to XSLT.
- Familiarize yourself with the basics of XML with Doug Tidwell's "Introduction to XML," a perennial favorite here on *developerWorks* (August 2002).
- Unfortunately, Schematron is in a bit of flux as it makes its way toward ISO standardization. For now, the easiest way to approach the specification is to start with the Schematron 1.5 spec and then read the update document for the 1.6 and ISO variants. If you prefer, you can read the ISO draft standard [PDF] directly, but it is very terse and formal.
- Find more XML resources on the *developerWorks* XML zone.
- Finally, find out how you can become an IBM Certified Developer in XML and related technologies.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- To work with the example files for this tutorial, download x-schematron-files.zip.
- Take a closer look at EXSLT, a community initiative to provide extensions to XSLT.
- Have a look around the Schematron home page and resource directory.
- When writing this tutorial, the author used the Scimitar implementation of ISO Schematron. A couple of other implementations of ISO Schematron are mentioned in this Weblog entry by Schematron inventor Rick Jelliffe.
- Build your next development project with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Join the Schematron mailing list for further Schematron news, information, and discussion.

About the author

Uche Ogbuji

Uche Ogbuji is a consultant and co-founder of Fourthought Inc., a software vendor and consultancy specializing in XML solutions for enterprise knowledge management. Fourthought develops 4Suite, an open source platform for XML, RDF, and knowledge-management applications. Mr. Ogbuji is also a lead developer of the Versa RDF query language. He is a computer engineer and writer born in Nigeria, living and working in Boulder, Colorado, USA. You can contact Mr. Ogbuji at uche.ogbuji@fourthought.com.