


<oo> → <dh> Digital humanities

Maintained by: David J. Birnbaum (djbitt@gmail.com) 

Last modified: 2017-04-10T13:50:02+0000

Validating references with Schematron

The text

This activity uses Skyrim (<http://dh.obdurodon.org/skyrim.xml>), a small text originally prepared by a former participant in our course. The file has a **<cast>** element at the top that contains, as child element, a list of *characters* (**<character>** elements) and *factions* (**<faction>** elements) that are mentioned in the **<body>** section, below the **<cast>** element. For this activity we are going to ignore characters initially and concentrate only on factions.

An entry for a faction in the cast list looks like:

```
<faction id="MythicDawn" alignment="evil"/>
```

Meanwhile, a reference to a faction in the body looks like:

```
The <faction ref="MythicDawn">assassins</faction> first attacked ...
```

Note that we use the **<faction>** element differently inside the **<cast>** element (where it has a unique **@id** attribute, plus other attributes we will ignore for now) and inside the **<body>** element (where it has a **@ref** attribute). Any mention of a **<faction>** element in the body must point to a matching **<faction>** element in the cast list. The way the pointing happens is that a **<faction>** element in the body always has a **@ref** attribute, and the value of that **@ref** attribute should point to (= match the value of) the **@id** attribute of some **<faction>** element in the cast list. In other words, 1) there should be no **<faction>** element in the body that does not point to a **<faction>** element in the cast list, and 2) there should be no **<faction>** element in the cast list that is not pointed to by at least one **<faction>** element in the body. The attribute that does the pointing is the **@ref** on the **<faction>** element in the **<body>**; the target of the pointing is an **@id** attribute on a **<faction>** element in the cast list.

You can assume that the developer has used a Relax NG schema and declared that the **@id** attribute is of type **xsd:ID**, that is, that it is an XML id. That means that it has certain properties, including constraints on the characters that it can contain and the fact that it is unique in the document. You can also assume that your Relax NG schema verifies that all **<faction>** elements in the cast list have an **@id** attribute and all **<faction>** elements in the **<body>** have a **@ref** attribute.

How a developer could screw up

There are at least two ways a developer could mangle these cross-references:

1. It's possible to encode a **<faction>** element in the body with a **@ref** attribute that doesn't point to (that is, correspond to) the **@id** attribute of a **<faction>** element in the cast list. For example, the **@id** attribute might be on a **<character>** element in the cast list, instead of on a **<faction>** element, or there might be no corresponding **@id** attribute at all.
2. It's possible to encode an unused **<faction>** element in the cast list, that is, one that is not pointed to by the **@ref** attribute of any **<faction>** element in the body. Since the inventory of factions in the cast list is supposed to summarize which factions occur in the body, such an error would bring the list out of sync with the reality of the body.

The task

We want to write a Schematron schema that will guard against the types of error described above by checking for consistency in two ways:

1. We want to write a rule for **<faction>** elements in the cast list to verify that all factions mentioned there also occur in the body. That is, there should be no faction listed in the cast list that is not also present in the body.
2. We want to write a rule for **<faction>** elements in the body that verifies that they have a **@ref** attribute that points to an **@id** attribute on a **<faction>** element in the cast list. Note that it isn't enough to check for the existence of a corresponding **@id** attribute, since there are **@id** attributes on **<character>** elements in the cast list, and not only on **<faction>** elements. Not only must the **@id** exist, but it must be associated specifically with a **<faction>** element in the cast list, and not with a **<character>** element.

Our solution

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process"
  xmlns="http://purl.oclc.org/dsdl/schematron">
  <pattern>
    <let name="cast-factions" value="//cast/faction/@id"/>
    <let name="body-factions" value="//body//faction/@ref"/>
    <rule context="cast/faction">
      <assert test="@id = $body-factions">The @id "<value-of select="@id"/>" occurs in the cast
        list, but not in the body.</assert>
    </rule>
    <rule context="body//faction">
      <assert test="@ref = $cast-factions">The @ref "<value-of select="@ref"/>" occurs in the
        body, but not in the cast list.</assert>
    </rule>
  </pattern>
</schema>
```

We begin by setting some *convenience variables*. What we mean by convenience variable is that we don't have to use them (we could have put the XPath expressions directly in the **@test** attributes), but they make our code more legible. **\$cast-factions** is a sequence of all **@id** values for all **<faction>** elements in the cast list. **\$body-factions** is a sequence of all **@ref** attributes for all **<faction>** elements in the **<body>**. (We could have used **distinct-values()** to get rid of the duplicates in the list of **@ref** values, but they do no harm in the tests we're running, so we just left them in. There cannot be any duplicates in the list of **@id** values because we have declared them as type **xsd:ID** in our Relax NG schema.)

The first rule fires on each **<faction>** element in the cast list. It checks whether the **@id** attribute value for that element matches the value of any of the **@ref** attributes on **<faction>** elements in the **<body>**. If not, it reports an error. To make the report more informative, we use the Schematron element **<value-of>** to print the offending value.

We use *general equals* (**=**) for this test, rather than *value comparison* (**eq**). General equals takes two sequences of any length and tests whether any member of one is a member of the other. If so, the test succeeds—no matter how many members of the sequence don't match! We have a sequence of one item on the left (the **@id** of the **<faction>** in the cast list that we're testing at the moment, and it qualifies as a sequence in the XPath sense even if it's a sequence of one item) and a sequence of many items on the right (all **@ref** values on all **<faction>** elements in the **<body>**), this test will succeed whenever the **@id** we're examining has a matching **@ref**. This type of one-to-many general comparison is very common in digital humanities coding. (Value comparison with **eq** does only one-to-one comparison, so if you try **eq** here, you'll get an error message because there is a sequence of more than one item on the right side.)

The second rule does the reverse. It fires on every **<faction>** element in the **<body>** and checks whether the value of the **@ref** attribute matches the value of an **@id** attribute on a **<faction>** element in the cast list.

An alternative that works

We could have hung the rules on the **@id** and **@ref** attribute values instead of on the **<faction>** elements that are their parents (making the necessary modifications to the paths in the code), and there's no particular reason to favor one of these strategies over the other.

Alternatives that don't work

It's possible to write rules that fire on **<cast>** or even on **<skyrim>** and that run one check of the entire document. This is much harder to code, although it can be done, but it's also less informative, since it doesn't associate the error message with a specific offending element. Even if you manage to poke the offending value into the error report, the red squiggly line will show up on **<cast>** or **<skyrim>**, so you'll have to work a bit harder to find the element you need to fix.

Some students in past semesters tried to run a single, global test on **<cast>** or **<skyrim>** just to count the number of **@id** values on **<faction>** elements in the cast list and compare that number to the count of distinct values of **@ref** attributes on **<faction>** elements in the **<body>**. That's not a tenable strategy because if, say, the factions in the cast list are "A", "B", and "C" and the ones in the **<body>** are "X", "Y", and "Z", there are three of each, but they don't correspond, you would want that to be reported as an error, and you can't do that just by counting them.

So how about **<character>** elements?

One might think one could use the same type of validation to check for cross-references on **<character>** elements: is every character mentioned in the cast list also encountered in the **<body>** and does every character mentioned in the **<body>** have a **@ref** attribute that points to the **@id** attribute of a **<character>** element in the cast list? This turns out to be harder than with factions because there are elements in the body like:

```
... the <character ref="hero Jauffre MartinSeptim">three of them</character> made their way ...
```

The problem here is that there is no **<character>** element in the cast list with an **@id** attribute whose value is "hero Jauffre MartinSeptim". Instead, this is a pointer to three separate characters in the header. The strategy for checking coreference therefore has to involve breaking apart the **@ref** attribute and checking each of the three pointers separately. This is the sort of task for which the XPath **tokenize()** function was created.

There is a hypothetical parallel problem concerning the other half of the assignment. Suppose there is a **<character id="Alex" loyalty="empire" alignment="neutral"/>** element in the head, but the only time Alex occurs in the body is in combination with another character, e.g., **<character ref="Alex Alathia">**. We can't just check whether there is a **@ref** attribute in the body that matches the string "Alex" because there isn't; here, too, we have to break apart the value of the **@ref** and check each part separately. This situation doesn't happen to occur in our text, but it is potentially possible and therefore something against which a well-designed development environment would protect the user.

Our extended solution

To avoid cluttering the screen, the Schematron below checks only **<character>** elements. In real life, you'd combine it with the one above.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt2"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process"
  xmlns="http://purl.oclc.org/dsdl/schematron">
  <pattern>
    <let name="cast-characters" value="//cast/character/@id"/>
    <let name="body-characters" value="//body//character/@ref"/>
    <rule context="cast/character">
      <assert test="@id = $body-characters">The @id "<value-of select="@id"/>"
        occurs in the cast list, but not in the body.</assert>
    </rule>
    <rule context="body//character">
      <assert test="@ref">The character "<value-of select="."/>" is missing its @ref
        attribute.</assert>
      <assert
        test="every $i in tokenize(normalize-space(@ref),'\s+') satisfies $i = $cast-characters"
        >The @ref "<value-of select="@ref"/>" occurs in the body, but not in the cast
        list.</assert>
      </rule>
    </pattern>
  </schema>
```

In our first rule, which fires on **<character>** elements that are children of **<cast>**, we check the **@id** attribute on every **<character>** element in the cast list to verify that it is pointed to by at least one **@ref** attribute on a **<character>** element in the **<body>**

Our second rule verifies that every **<character>** element in the **<body>** has a **@ref** attribute. The developer could have checked this in Relax NG by making the **@ref** attribute obligatory, but she didn't. To our surprise, although we had used this document for other exercises previously, until we wrote this Schematron rule we had never noticed that there are two **<character>** elements in the **<body>** that don't have any **@ref** attribute! This is real inadvertent error, and had we already learned Schematron when the original developer encoded this file, she would have been able to use it to catch this error and fix it.

Once we've confirmed that there is a **@ref** attribute on the **<character>** element in the **<body>** that we're looking at at the moment, we use the XPath **tokenize()** function to break it apart into pieces. We then use the **every X in Y satisfies** construction (Kay, p. 646 ff.) to check each one individually.

Why not use ID/IDREF validation?

The Relax NG **xsd:ID** datatype is guaranteed to be unique in the document, and it is also guaranteed to conform to the lexical specification (= spelling rules) for an *XML non-colonized name*, abbreviated NCName. NCNames must begin with an alphabetic character and can otherwise contain alphanumeric characters and selected punctuation. They cannot contain most punctuation characters and they cannot contain whitespace characters. If you declare an attribute as being of type **xsd:ID** in your schema and try to use an illegal character, <oXygen/> will notify you of the error. You can read a more precise, human-readable description of what is and is not permitted in an NCName (and therefore in an attribute value of type **xsd:ID**) at <http://stackoverflow.com/questions/1631396/what-is-an-xsncname-type-and-when-should-it-be-used>.

Attributes declared as the Relax NG datatype **xsd:IDREF** must have values that match an item of type **xsd:ID** in the same document. This means that they have the same requirements about legal and illegal characters, and they also have to match (that is, refer to) a real declared **xsd:ID** value in the same document. There is also an **xsd:IDREFS** datatype which refers to a white-space-delimited set of one or more **xsd:ID** values, which means, for example, that you can tag the word "they" in the body of your text and have it refer to the Three Stooges with:

```
... and then <character ref="curly larry moe">they</character> said ...
```

In the preceding example, if the **@ref** attribute were declared as having type **xsd:IDREFS**, <Oxygen/> would verify that there were values of type **xsd:ID** for “curly”, “larry”, and “moe” in the document, and it would report an error if it couldn’t find all three.

The limitation of ID/IDREF is that it can only compare an **xsd:IDREF** value to all **xsl:ID** values in the document. This imposes two more specific limitations on its utility:

- A parser can confirm that, say, a **@ref** attribute on a **<faction>** element in the **<body>** points to an **xsd:ID** somewhere, it cannot determine whether the **xsd:ID** is specifically on a **<faction>** element in the **<cast>**.
- A parser cannot check an **xsd:IDREF** attribute value in one document against an **xsd:ID** attribute value in a different document. ID/IDREF valuation happens only within a single document.

The Schematron strategy illustrated above does not suffer from either of these limitations, and this example illustrates how it overcomes the first of them.