newtFire {dhlds}
Authored by: Nicole L. Lottig (nll29 at pitt.edu) and Brooke A. Stewart (bas160 at pitt.edu) Edited and maintained by: Elisa E. Beshero-Bondar (ebb8 at pitt.edu)

Last modified: Sunday, 06-Aug-2017 18:16:15 EDT. Powered by firebellies.

Schematron Exercise 2

Preliminaries

To work on this assignment, you will need to to find and do the following:

- **Information resources at the ready:** Review our Schematron tutorial, and read more about the XPath functions and syntax we describe below either on the web (see w3Schools' "XSLT, XPath, and XQuery Functions", Obdurodon's "The XPath Functions We Use the Most") or through offline searching with the index of the Michael Kay book.
- **XML file to test:** Save this TEI file locally and open it in <oXygen/>: Emily Dickinson's Fascicle 16 poems. You will need to associate your Schematron file with this document **in addition to** the currently associated TEI schema lines.
- Open a new Schematron document in <oXygen/> by going to **File → New** and typing "Schematron" in the "Type filter text" box, or by going to **File → New → New Document → (scroll to Schematron in the alphabetized list) → Schematron**. Once opened, you will keep the default xml line at the top, but you will delete everything from `<sch:schema>` down. To write Schematron rules for a document in the TEI namespace, you will then replace this with:

```
<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">
        <ns uri="http://www.tei-c.org/ns/1.0" prefix="tei"/>

    </schema>
```

- Write your Schematron patterns **inside** the `</schema>` root element.
- Use the `tei:` prefix before each of your elements since we are working with a document in the TEI namespace. Remember that we do **not** use that prefix before attributes because attributes are in no namespace.

Analysis of the task

The goal:

The Dickinson project team is using TEI `<app>` elements inside the lines of Dickinson's poems when they need to encode a set of *variant* words or phrases that appear in different publications, labeled in the `<rdg>` elements with their `@wit` attributes. You are working with a single file representing a set of poems from a collection of manuscripts or *fascicle* that Emily Dickinson bundled and bound together herself. For this assignment, you will write Schematron to function on top of the established TEI Relax NG Schema to help ensure that the `<app>` and `<rdg>` elements are written properly according to the rules of the team. You will need to write a few rules to make sure that particular elements and attributes are appearing where we need them to, to make sure the poems are appearing in the proper order in this document (Poem 1 through Poem 11), and to control for missing or additional white spaces around our tags that might be distorting our representation of the poems.

Creating the rules step by step:

1. **Make sure each `<rdg>` element has no other attribute but `@wit`.** We want to make sure that the `<rdg>` element has nothing but an `@wit` attribute, but it *must* have this attribute. (The TEI schema by itself will allow other attributes or no attributes at all on this element, but we want to make sure that our project team only uses just this `@wit` attribute and not others.) Consider that we want the Schematron to tell us when `@wit` is *missing* and decide whether you need to write an `<assert>` or a `<report>` rule for this. Our solution uses the `not()` function in `@test` to fire if the `<rdg>` does not have `@wit` (including if it has any attribute other than `@wit`).

   Remember to use the `tei:` prefix before your element names! (Examples: `tei:app` and `tei:rdg`)

2. **Make sure there are one or more `<rdg>` elements inside an `<app>` element.** For this rule we want to check that an `<app>` element has a *count* of at least one or more than one `<rdg>` element. **Note:** For every new rule that matches in some way on the `<rdg>` context, you need to position it inside a new Schematron `<pattern>` element because otherwise only the first rule at a given context will fire and the others will remain passive.

3. **Make sure all of the poems are in the correct counting order within the document.** The team used XSLT to combine eleven separate documents, each holding a single poem, into the one XML file you are working with to hold the entire collection. However, the poems may not have transferred over in the correct order. For example, maybe Poem 6 comes after Poem 7 instead of coming directly after Poem 5. The rule that we will create now will help to check if the poems are in order so you can rearrange them if they are not. For this, we need to do the following:

    - First we need to look over our XML document that is holding all of the poems and find out *where* all of the poem titles are located in the hierarchy. Those poem titles each hold a number (1 to 11) that indicates where they properly sit in the sequence of the collection. We know that these `<title>` elements are positioned inside the `div/head` of each poem. Notice that each begins with the same pattern of text: the word "Poem" followed by a white space and a one or two-digit number.

    - Think about what we need to do: We want to make sure that these poems are in the correct order, based on the number given in their title. Is the poem titled Poem 2 immediately following the one titled Poem 1? If we look ahead and evaluate each poem's title in relation to the one that follows it, we only want to look at poems *that are followed by another poem*. (The last poem will not have a poem following after it to compare, but it will already be worked into the test because it will be the poem following the second-to-last poem.) Write your Schematron `<rule>` element and set its `@context` accordingly. (Our solution sets the `@context` at the `title` position. Think about whether you want to use the `following-sibling::` or the `following::` axis. Either way, you will need to compare a number in the `<title>` element of a current poem to that of the **first**, immediately-following poem.)
    - Decide whether you want to write an `<assert>` or `<report>` test that isolates the number of the poem inside the title element at your context. Our solution uses the `number()` function to convert the numeral(s) into a literal number, and then adds `+ 1` to test if that value equals the number of the poem given in the next following poem `div` (stepping down into *its* `title` element to isolate and convert and read *its* number). (Think about why we need to add **+ 1** here, or perhaps alternative ways you could write this test.)
    - You need to isolate just the number after the word “Poem” in the title, and to do this you need the `substring()` function (which you may wish to look up in Michael Kay or w3schools to see how this is formatted). The `substring()` function takes three arguments. The first argument indicates the XPath node (so if you set your rule context at the title, you would just invoke the `self::*` or dot(`.`) as the first argument). The second and third arguments are numbers: The second argument gives the numerical position of the character in the whole string of text that indicates the point where you want to start extracting your substring (so for this, count over from the start of the title to the first digit you want). The third argument indicates *how many* characters you want to extract into your substring. So the function is set up like this:

      ```
      substring(XPath, character-position-number-to-start, number-of-characters-to-extract)
      ```

      Note: since we have 11 poems, we are going to need to extract two characters to deal with Poem 10 and Poem 11.
    - Wrap your `substring()` in a `number()` function to convert it, and now work with it *as* a number. Add `+ 1` to it, and see if that value, `(substring() + 1)` equals the `substring()` in the title of *just* the very next poem in the sequence.
    - Test your rule. Our file is deliberately out of sequence, so you can expect to see errors if your rule is firing correctly.
4. **Test the values of the `@wit` attributes sitting on the `rdg` elements to be sure they are not mistyped.** This is something you are likely to need in your projects, so we direct you to our special Schematron tutorial on testing unique identifiers, which shows you how to work with `@xml:id`s (unique identifiers) and their corresponding referencing attributes. Can you adapt the code in our tutorial to work with this file and its positioning of the list of witnesses in this document?

5. **Optional Challenge: Control the white space around the <app> and <rdg> elements in a line of poetry.** As the team works on coding these poems, it is very easy for them to accidentally remove or add white space in applying <app> and <rdg> elements. It is very easy to make two words run together by accident, for example, by coding like this:

```
<l n="1">When we stand on the tops of<app>
 <rdg wit="#df16">Things—</rdg>
 <rdg wit="#bm">things</rdg>
</app>
</l>
```

Notice that there is no space before the opening <app> tag and no space inside the opening <rdg> tag, so when the team transformed this to view the first witness in HTML, we saw something like this:
When we stand on the tops ofThings—
To deal with this, we need to recognize that sometimes we want a white space in between the main line of text and the starting <app> element, and sometimes we do not.

- We **do not want to add a space** when the line of text before <app> ends with white space already, when it has a special punctuation mark, a dash (–) or a quotation mark (`&quot;`) designed to connect with the text in the <rdg> element(s).
- We need to **add** a white space whenever the line of text before <app> ends with something *other than* the three characters we described above *and* the `rdg` element inside begins with a letter (another alphabet character).
- We might have to **remove** an extra white space when the line of text before <app> ends with *any non-space character followed by a space*, and the <rdg> element opens with a white space.
- You may also want to test for white space at the end of an rdg element when its `parent::app` is *followed by* a string of text.

For our purposes, if you can write a Schematron rule that addresses even just one of the above scenarios, that is sufficient, though we hope that if you succeed with one test, you will figure out how to write one or two others! To control for white space, we created a pattern with a single rule set on the `@context` of the `tei:rdg` element, because we need to look at each <rdg> element in turn to see if we have a white space problem, and there are often multiple <rdg> elements inside each line. When we set the context to the whole line of poetry, it might have multiple sets of <app> elements inside, and we cannot write a precise enough rule to address the spans of text we need. To proceed, we need to understand something about **mixed content**: When an element like the TEI `l` (for a line of poetry) contains a mixture of `text()` and other elements, the `text()` node is sitting in a **sibling** relationship to the nested elements, so that a span of text in a line of a Dickinson poem is sitting on the `preceding-sibling::` axis in relation to the <app> element that follows it. If you write your rule as we did, from the context of the <rdg> elements, you will need to write your test to reach up to to the parent <app> elements and walk over to the `preceding-sibling::text()` node. We use the `matches()` function in our Schematron `@test` because it works with regular expressions and helps us to identify the particular conditions we are looking for. (Look up this function in one of the sources we list in the Preliminaries section of this assignment to be sure you understand how to write it.) Specifically, we are going to need a *two-part test*, and we can use the `matches()` function twice, joined by the word `and` to see first if a) the line of text that is the first preceding-sibling of our parent <app> *ends with* something in a regex character set, **and** b) the contents of our context <rdg> element *starts with* something in a regex character set, like this:

```
test="matches(. . .) and matches(. . .)"
```

or

```
test="not(matches(. . .) and matches(. . .)"
```

or some combination of these.

Note that the `matches()` function takes two arguments like this: `matches(Xpath-location, 'regex-pattern')`. You might be wondering why we aren't using the functions `starts-with()` or `ends-with()`. The answer is that these do not help us with finding regular expressions, but `matches()` can look for a regex pattern *wherever we need it*. To designate the **start of a line** in regex (or the start of the text in a given XPath node, use the regex caret, `^`, at the start of the regex pattern you are hunting for, and to designate **the end of the text**, use the regex dollar sign, `$` at the end of your regex pattern.

Bonus task: You will likely have difficulty with matching on a quotation mark, because if you try to include it literally in the character set (or even escape it), it will be interpreted as the end of the schematron attribute and will result in a formedness error, munging your Schematron code. Consider it a bonus task on this assignment to find a way to match on a straight quotaton mark. Hint: you will need to escape the literal quotation mark using `&quot;`, but you won't be able to include it in a `[ ]` character set.

See how far you can get with this Optional Challenge Task and if you get stuck, record what you tried and what didn't work. Do your tests fire? You should see some white space errors in the file as we presented it, but you should also tinker with the white space just before an `<app>` tag and at the start of an `<rdg>` element.

Submission

Upload your completed Schematron schema AND the Dickinson poems XML ***with your Schematron associated*** to Courseweb, and follow our standard filenaming conventions for homework assignments uploaded to Courseweb.