


<oo> → <dh> Digital humanities

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

Last modified: 2018-03-12T15:41:00+0000

XSLT assignment #3

The input text

For this assignment we'll continue to use <http://dh.obdudon.org/skyrim.xml>. You should right-click on this link, download the file, and open it in <Oxygen/>. You don't need the Relax NG schema, but if you'd like to look at it, it's available at <http://dh.obdurodon.org/skyrim.rnc>.

Overview of the assignment

For this assignment we're going to work with the <body> element, concentrating on processing the in-line elements to style the text. You can use some of the basic HTML in-line elements, like for emphasis or for strong emphasis, but you'll also want to use CSS to set some elements to different colors or background colors or borders or fonts or font sizes or font styles (e.g., italic) or font weights (e.g., bold) or text decoration (e.g., underlining) or text transformation (e.g., convert to all upper case) ... well ... anything else. We describe below how to do that.

There are six types of in-line elements in the input XML document:

- <QuestEvent>
- <QuestItem>
- <character>
- <epithet>
- <faction>
- <location>

Some are immediately inside a <paragraph> and some are inside other elements that are inside paragraphs. You may not know at the outset which ones can be inside which other ones, or how deeply they can nest. Happily, with XSLT, unlike with many other programming languages, you don't need to care about those questions!

How to process richly mixed content

Prose paragraphs with in-line elements that might contain other in-line elements are richly mixed content, with varied and unpredictable combinations of elements and plain text. This is the problem that XSLT was designed to solve. With a traditional procedural programming language, you'd have to write rules like "inside this paragraph, if there's a <QuestEvent> do X, and, oh, by the way, check whether there's a <QuestItem> or a <location> inside the <QuestEvent>, etc." That is, most programming languages have to tell you what to look for at every step. The elegance of XSLT when dealing with this type of data is that all you have to say inside paragraphs and other elements is "I'm not worried about what I'll find here; just process (apply templates to) all my children, whatever they might be."

The way to deal with mixed content in XSLT is to have a template rule for every element and use it to output whatever HTML markup you want for that element and then, inside that markup, to include a general `<xsl:apply-templates/>`, not specifying a `@select` attribute. For example, if you want your `<QuestEvent>` to be tagged with the HTML `` tags, which means “strong emphasis” and which is usually rendered in bold, you could have a template rule like:

```
<xsl:template match="QuestEvent">
  <strong>
    <xsl:apply-templates/>
  </strong>
</xsl:template>
```

You don’t know or care whether `<QuestEvent>` has any child nodes or, if it does, what they are. Whatever they are, this rule tells the system to try to process them, and as long as there’s a template rule for them, they’ll get taken care of properly somewhere else in the stylesheet. If there are no child nodes, the `<xsl:apply-templates/>` will apply vacuously and harmlessly. As long as every element tells you to process its children, you’ll work your way down through the hierarchy of the paragraph without having to know which elements can contain which other elements or text nodes.

Taking stock: when to use `@select`

In an earlier XSLT assignment, where you built HTML tables of characters and factions, you used `<xsl:apply-templates select="...">`, specifying exactly what you wanted to process where. That makes sense when your input (the `<character>` and `<faction>` elements inside the `<cast>` element at the beginning of the document) and output (an HTML table) are very regular in structure. *Use the `@select` attribute when you know exactly what you’re looking for and where you want to put it.*

In this assignment, on the other hand, you don’t know (and don’t need to know) the order and nesting hierarchy of whatever salad of elements and plain text you might find inside a paragraph or its subelements. You just want to process whatever comes up whenever it comes up. `<xsl:apply-templates/>` without the `@select` attribute says “apply templates to whatever you find.” *Omit the `@select` attribute where you don’t want to have to think about and cater to every alternative individually.* (You can still treat them all differently because you’ll have different template rules to “catch” them, but when you assert that they should be processed, you don’t have to know what they actually are.)

What should the output look like

HTML provides a limited number of elements for styling in-line text, which you can read about at http://www.w3schools.com/html/html_formatting.asp. You can use any of these in your output, but note that presentational elements, the kind that describe how text looks (e.g., `<i>` for “italic”), are generally regarded as less useful than descriptive tags, which describe what text means (e.g., `` for “emphasis”). Both of the preceding are normally rendered in italics in the browser, but the semantic tag is more consistent with the spirit of XML than the presentational one.

The web would be a dull world if the only styling available were the handful of presentational tags available in vanilla HTML. In addition to those options, there are also ways to assign arbitrary style to a snippet of in-line text, changing fonts or colors or other features in mid-stream. To do that:

1. Before you read any further in this page, read our [Using `` and `@class` to style your HTML page.](#)

2. To use the strategies described at that page, create an XSLT template rule that transforms the element you want to style to an HTML `` element with a `@class` attribute. For example, you might transform `<faction ref="MythicDawn">assassins</faction>` in the input XML to `assassins` in the output HTML. You can then specify CSS styling by reference to the `@class` attribute, as described in the page we link to above.
 - Note that you can make your transformations very specific. For example, instead of setting all `<faction>` elements to the same HTML `@class`, you can create separate template rules to match on factions according to their attribute values. For example, `<xsl:template match="faction[@ref='MythicDawn']">` is a normal XPath expression to match `<faction>` elements only if they have a `@ref` attribute with the value "MythicDawn".
 - If you really want to exercise your XPath skills, note that in the header some factions are described (with an `@alignment` attribute) as "evil", "good", or "neutral". You can write a matching rule that will *dereference* the `@ref` attribute on, say, `<faction ref="MythicDawn">assassins</faction>`, look up whether this is an evil, good, or neutral faction, and set the `@class` value accordingly. You could make all good factions one color and all evil factions a different color, letting XPath look up the moral alignment of a faction for you.
3. Setting the `@class` attributes in the output HTML makes it possible to style the various `` elements differently according to the value of those attributes, but you need to create a CSS stylesheet to do that. Create the stylesheet (just as you've created CSS in the past), and specify how you want to style your `` elements. Link the CSS stylesheet to the HTML you are outputting by creating the appropriate `<link>` element in your output HTML (you can remind yourself how to do that at the bottom of our [Introduction to CSS](#)).

When the smoke clears

What you should produce, then, is:

- An XSLT stylesheet that transforms the `<body>` element and its contents into HTML.
- The resulting HTML should style the six types of in-line elements listed above. At least some of those styles should be set using `` elements with the `@class` attribute.
- You need to create a CSS file, linked to your output HTML, that specifies how to style the output document. You can look up the most useful of the available CSS properties at <http://www.w3schools.com/css/>. We'd suggest following the links on the left under "CSS styling" for styling backgrounds, text, and fonts, as well as the link for borders under "CSS box model".

Please upload your XSLT, HTML, and CSS files to CourseWeb.