


<oo> → <dh> Digital humanities

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

Last modified: 2018-02-28T14:12:07+0000

XSLT assignment #1

The assignment

Your assignment is to create an XSLT stylesheet that will transform *Bad Hamlet* into a hierarchical outline of the titles of acts and scenes in HTML. This isn't very interesting on its own, of course, but if you were transforming the entire document into HTML for publication on the web, this might serve as the skeleton. It might also stand on its own as a table of contents at the top of such a publication, so that the reader could click on the title of a scene to jump to that location in the file.

If you're feeling adventurous, you're welcome to include more information, whether of a publication-oriented sort (e.g., speakers, speeches, stage directions, etc., as if you were publishing the entire play) or as a foray into exploration and analysis (e.g., list of characters who speak in each scene, perhaps with a count of their speeches, length of speeches, etc). The only required content of your homework, though, is the HTML outline of act and title chapters, which might look something like:

- Act 1
 - Act 1, Scene 1
 - Act 1, Scene 2
 - Act 1, Scene 3
 - Act 1, Scene 4
 - Act 1, Scene 5
- Act 2
 - Act 2, Scene 1
 - Act 2, Scene 2
- Act 3
 - Act 3, Scene 1
 - Act 3, Scene 2
 - Act 3, Scene 3
 - Act 3, Scene 4
- Act 4
 - Act 4, Scene 1
 - Act 4, Scene 2
 - Act 4, Scene 3
 - Act 4, Scene 4
 - Act 4, Scene 5
 - Act 4, Scene 6
 - Act 4, Scene 7
- Act 5
 - Act 5, Scene 1
 - Act 5, Scene 2

The underlying HTML, which we generated using XSLT, is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  SYSTEM "about:legacy-compat">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hamlet</title>
  </head>
  <body>
    <ul>
      <li>Act 1
        <ul>
          <li>Act 1, Scene 1</li>
          <li>Act 1, Scene 2</li>
          <li>Act 1, Scene 3</li>
          <li>Act 1, Scene 4</li>
          <li>Act 1, Scene 5</li>
        </ul>
      </li>
      <li>Act 2
        <ul>
          <li>Act 2, Scene 1</li>
          <li>Act 2, Scene 2</li>
        </ul>
      </li>
      <li>Act 3
        <ul>
          <li>Act 3, Scene 1</li>
          <li>Act 3, Scene 2</li>
          <li>Act 3, Scene 3</li>
          <li>Act 3, Scene 4</li>
        </ul>
      </li>
      <li>Act 4
        <ul>
          <li>Act 4, Scene 1</li>
          <li>Act 4, Scene 2</li>
          <li>Act 4, Scene 3</li>
          <li>Act 4, Scene 4</li>
          <li>Act 4, Scene 5</li>
          <li>Act 4, Scene 6</li>
          <li>Act 4, Scene 7</li>
        </ul>
      </li>
      <li>Act 5
        <ul>
          <li>Act 5, Scene 1</li>
          <li>Act 5, Scene 2</li>
        </ul>
      </li>
    </ul>
  </body>
</html>
```

The DOCTYPE declaration at the top (**<!DOCTYPE html SYSTEM "about:legacy-compat">**) is a schema declaration. It is similar to the declaration that <oxyen/> creates when you author a new XHTML document by hand (**<!DOCTYPE html>**), except that, for obscure reasons that aren't very interesting, it takes

a different form when you generate it as part of an XSLT transformation. You can think of it as a magic incantation, and we explain below how to invoke it.

We've used HTML unordered lists (``) elements. The only content allowed inside a `` element is list items (``), and we've nested them, so that each list item that represents an act contains the title of that act followed by an embedded `` that contains, in turn, a separate list item for the title of each scene. This isn't the only way to format this type of outline and you're welcome to take a different approach. For example, if you'd like to include the full text of the play, that is, the stage directions and speeches, the embedded list format isn't really appropriate. In that case, we would use the HTML header elements (`<h1>` through `<h6>`) to create hierarchical headers. You can read more about HTML lists and headers at <http://www.w3schools.com>.

Before you begin

Both your input document and your output documents have to be in the correct namespace, and you need to tell your XSLT stylesheet about that.

- **Input namespace:** Your input document, *Bad Hamlet*, is in the TEI namespace, which means that your XSLT stylesheet must include an instruction specifying that when it tries to match elements, it needs to match them in that namespace. You specify the namespace of the input document by creating a `@xpath-default-namespace` attribute on the `<xsl:stylesheet>` root element of your XSLT stylesheet and setting it equal to the `@xmlns` value of the input document. When you create a new XSLT document in <Oxygen/> it won't contain that instruction, so you need to add it (in blue below).
- To ensure that the output will be in the HTML namespace, which is required for any HTML web page, you need to specify a default output namespace declaration (in fuchsia below).

Finally, to output the required DOCTYPE declaration, we also create an `<xsl:output>` element as the first child of our root `<xsl:stylesheet>` element (in green below). Our modified skeleton looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0"
  xpath-default-namespace="http://www.tei-c.org/ns/1.0"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:output method="xml" indent="yes" doctype-system="about:legacy-compat"/>
</xsl:stylesheet>
```

Guide to approaching the problem

Our XSLT transformation (after all this housekeeping) has three template rules:

1. We have a template rule for the document node (`<xsl:template match="/">`), in which we create the basic HTML output: the `<html>` element, `<head>` and its contents, and `<body>`. Inside the `<body>` element that we're creating, we use `<xsl:apply-templates>` and select the acts (using an XPath expression as the value of the `@select` attribute).
2. We have a separate template rule that matches acts, so it will be invoked as a result of the preceding `<xsl:apply-templates>` instruction, and will fire once for each act. Inside that template rule we create a new list item (``) for the act being processed and inside the tags for that new list item we do two things. First, we apply templates to the `<head>` for the act, which will eventually cause its title to be output. Second, we create wrapper `` tags for the nested list that will contain the titles of the scenes. Inside that new `` element, we use an `<xsl:apply-templates>` rule to apply templates to (that is, to process) the scenes of that act.

3. We have a separate template rule that matches scenes, and that just applies templates to the **<head>** element in each scene, which ultimately causes the textual content of the **<head>** element to be output. This rule will fire once for each scene in the play, and it will be called separately for the scenes of each act, so that the scenes will be rendered properly under their acts.

We don't need a template rule for the **<head>** elements themselves because the built-in (default) template rule in XSLT for an *element* that doesn't have an explicit, specified rule is just to apply templates to its children. The only child of the **<head>** elements is a **text()** node, and the built-in rule for **text()** nodes is to output them literally. In other words, if you apply templates to **<head>** and you don't have a template rule that matches that element, ultimately the transformation will just output the textual content of the head, that is, the title that you want.

Important

- Those who like to read ahead or already have some programming experience with other languages may have noticed that XSLT includes an **<xsl:for-each>** instruction that *could* be used to solve this problem. *We are prohibiting its use for now*; your solution must use **<xsl:template>** and **<xsl:apply-templates>** rules instead. There's a Good Reason for this, which we'll explain later, when we talk about situations where you *should* use **<xsl:for-each>**.
- *Before submitting your homework, you must run the transformation* to make sure the results are what you expect them to be. There's a guide to running XSLT transformations inside <Oxygen/> at <http://dh.obdurodon.org/oxygen-xslt-configuration.html>. If you don't get the results you expect and can't figure out what you're doing wrong, you're welcome to post an inquiry to the discussion board. You can't just ask for the answer, though; you need to describe what you tried, what you expected, what you got, and what you think the problem is. We often find, just as we're preparing to post our own queries to programming discussion boards, that having to write up a description of the problem helps us think it through and solve it ourselves (the technical term for this phenomenon is **rubber duck debugging**). We're encouraging you to discuss the homework on the discussion boards because that's also helpful for the person who responds; we've found that answering someone else's inquiry and troubleshooting someone else's problem helps us clarify matters for ourselves.