




[newtFire {dhlds}](#)

Maintained by: Elisa E. Beshero-Bondar (ebb8 at pitt.edu)  Last modified: Tuesday, 17-Oct-2017 15:26:37 EDT. [Powered by firebellies](#).

XSLT Exercise 1

Our very first XSLT assignment is an Identity Transformation, a kind transformation we have to do frequently in our projects when we need to make specific changes to our encoding. We want to make some small changes in our Georg Forster file to make better choices of TEI elements for some of our tags.

To begin, download the Georg Forster file from here: [ForsterGeorgComplete.xml](#) and open it in <oXygen>. We don't want to change much about this file, but we do want to alter its tagging just a little, and that is a good occasion to write an Identity Transformation XSLT, converting our XML to XML that is meant to be (for the most part) *identical* to the original.

Here are two changes we want to make to our XML file:

- Looking through the file in the Outline view, we notice that our <head> elements inside each <div type="chapter"> are holding <l> elements, which we originally applied to preserve line breaks in the original document. But we really should not be using the <l> element, because in TEI that element is reserved for a line of poetry! We should change our tagging, and we think we should instead *end* each line with the self-closing <lb/> element used to record a (non-poetry) line-break in TEI.
- Scrolling through the document, we notice we have used <emph> elements in TEI when we wanted to indicate a rendering in italics in the original. Just like the problem with the use of <l>, that was a mistaken application of the TEI (even though it looks perfectly valid), because the <emph> element is only supposed to be used when a writer is placing *strong emphasis* on a word or phrase. In this document, the <emph> elements are being applied to designate non-English words and book titles, so this tagging is not really for emphasis. We really should be using the TEI <hi rend="italic"> tagging for these instead, since this element is designated for highlighting of any kind.

You may already be calculating how to do these tasks with a regular expression Find and Replace, and while we know you could do that, our purpose with this exercise is to make the changes using an XSLT transformation, and we hope you will learn some things about how XSLT works through this exercise!

To begin, open a new XSLT stylesheet in <oXygen> and switch to the XSLT view. We will have some housekeeping to do as we get started.

Namespaces matter! Setting up an XSLT stylesheet to Read TEI

Georg Forster's *A Voyage Round the World* is coded in the TEI namespace, which means that your XSLT stylesheet must include an instruction at the top to specify that when it tries to match elements, it needs to match them in that TEI namespace. When you create a new XSLT document in <oXygen/> it won't contain that instruction by default, so *whenever you are working with TEI* you need to add it (See the text in [blue](#) below). We also need to make sure that our XSLT parser understands it is outputting results to the TEI namespace, so we change one more line (See the text in [red](#) below). Our modified stylesheet template looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xpath-default-namespace="http://www.tei-c.org/ns/1.0"
  xmlns="http://www.tei-c.org/ns/1.0"
  version="3.0">

</xsl:stylesheet>
```

Writing the Identity Transformation!

1. We will give you your first template rule, to set this as an *identity transformation*. We're going to use a new form for this in version XSLT 3.0, so that is why we have set `version="3.0"` in our stylesheet template above. On future assignments we are setting the default version 2.0 for transforming to HTML mostly because the old version is better tested for processing HTML output, but for an identity transformation of XML to XML, we like the efficient new code we can write in version 3.0. (You can see an old form here in the first template rule of our Identity transformation of Shakespeare's sonnets, which you can download, save and open from [here](#). That old first rule matches on all nodes, elements and attributes throughout the document and simply copies them. It's perfectly fine to use that older template rule in place of the one we show you below, but we like the simplicity of this new form even better!).
`<xsl:mode on-no-match="shallow-copy"/>`

This XSLT statement is the *opposite* of the `xsl:template match` we have been showing you in [our XSLT tutorial](#). You basically say, if I do not write a template rule to *match* an element, attribute, or comment node, really of any part of the document that I do not mention in a template match rule, XSLT should simply make a copy of that element and output it. Try running this and look at your output: it will look exactly *identical* to the current XML document. Obviously we do not need to do this *unless* we want to make changes with template match rules! There is another way to copy, called "deep copy" in XSLT, but we do not want use it here. When you use "deep copy" in XSLT, you reproduce the full directory tree underneath a given element, so the understanding is that we would match on the root element *only*, and reproduce all the descendents of that one node just as they are. We like the "on-no-match-shallow-copy" approach because we do not necessarily want to copy every node just as it is in the original. We only want to copy if it we do not want to write a new template rule that will change it.

2. Next, we will simply write our template rules to match on the particular elements we wish to change. You may wish to start with the simpler of the two, to convert all the `<emph>` elements into `<hi rend="italics">` in the output XML. Review our [Introduction to XSLT](#) to see how to write a template match on any particular element, and how to output as a different element in its place using `<xsl:apply-templates/>`.
3. Now, write the template rule that will match *only* on `<l>` elements that are children of `<head>` elements. And see if you can figure out how to replace these by positioning the self-closing line-break element `<lb/>`, positioned in the correct spot in relation to `<xsl:apply-templates/>` so that the `<lb/>` sits at the end of a line.

4. When we write Identity Transformation XSLTs, we often work with **Attribute Value Templates** (or **AVT**), a handy special format in XSLT that helps us to add attributes to elements like `<p>` or `<l>`, and work with values we calculate. This is the tool we use when we want to tell the computer to count and calculate line or paragraph numbers to output in an attribute (like `@n` or `@number`). An AVT offers a special way to extract or calculate information from our input XML to output in an attribute value (for example, this lets us come up with a `count()` of where the particular line we are processing sits in relation to all the `preceding::` line elements ahead of it). You need to look at some examples of AVTs in order to write one yourself, so for this last task, go and look at the examples in Obdurodon's [Attribute Value Templates \(AVT\) tutorial](#). After reading the AVT tutorial, write **two more template rules** to add `@n` attributes that automatically number the `<div>` elements for Books, and the `<div>` elements for Chapters. (We would ask you to number the paragraphs, too, but we already did that!) **Hint:** For help with teaching the computer how to count these properly, look at my example ID-transform stylesheet that adds line numbers to a series of sonnets, downloadable from [here](#) if you didn't download it earlier from the Introduction to XSLT tutorial.) We will return to this later, since you will be working with AVTs in later XSLT exercises and almost certainly in your projects.

When you are finished, save your XSLT file and your XML output of the Georg Forster file, following our usual [homework file naming conventions](#), and upload these to the appropriate place in Courseweb.