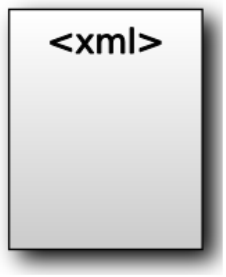
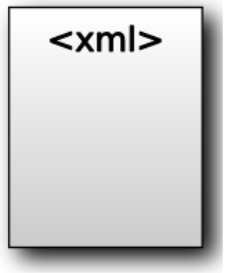
XSLT: where does it fit in?

Martin Holmes

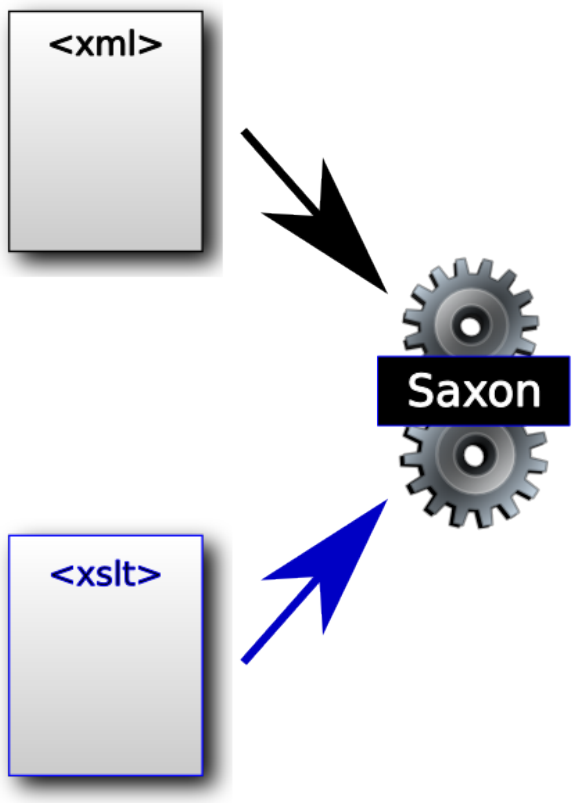
XSLT in the publishing workflow



XSLT in the publishing workflow

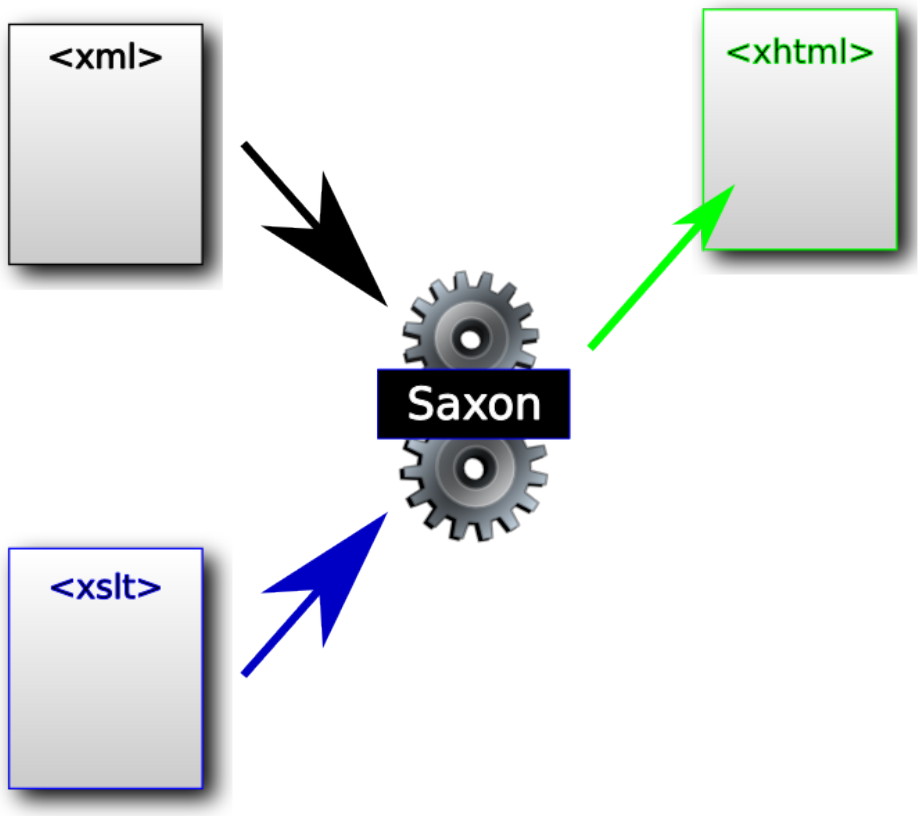


XSLT in the publishing workflow

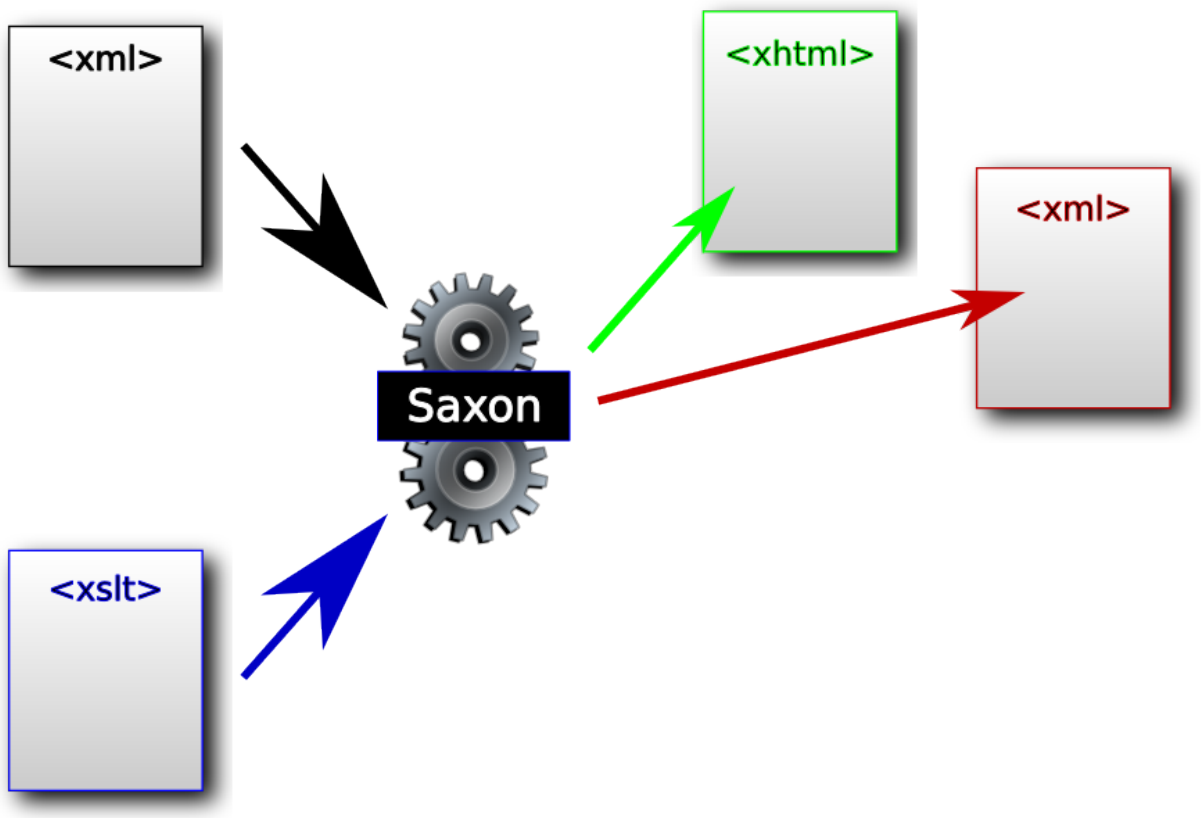


Elicit some of the possible outputs we might produce from XML using XSLT.

XSLT in the publishing workflow

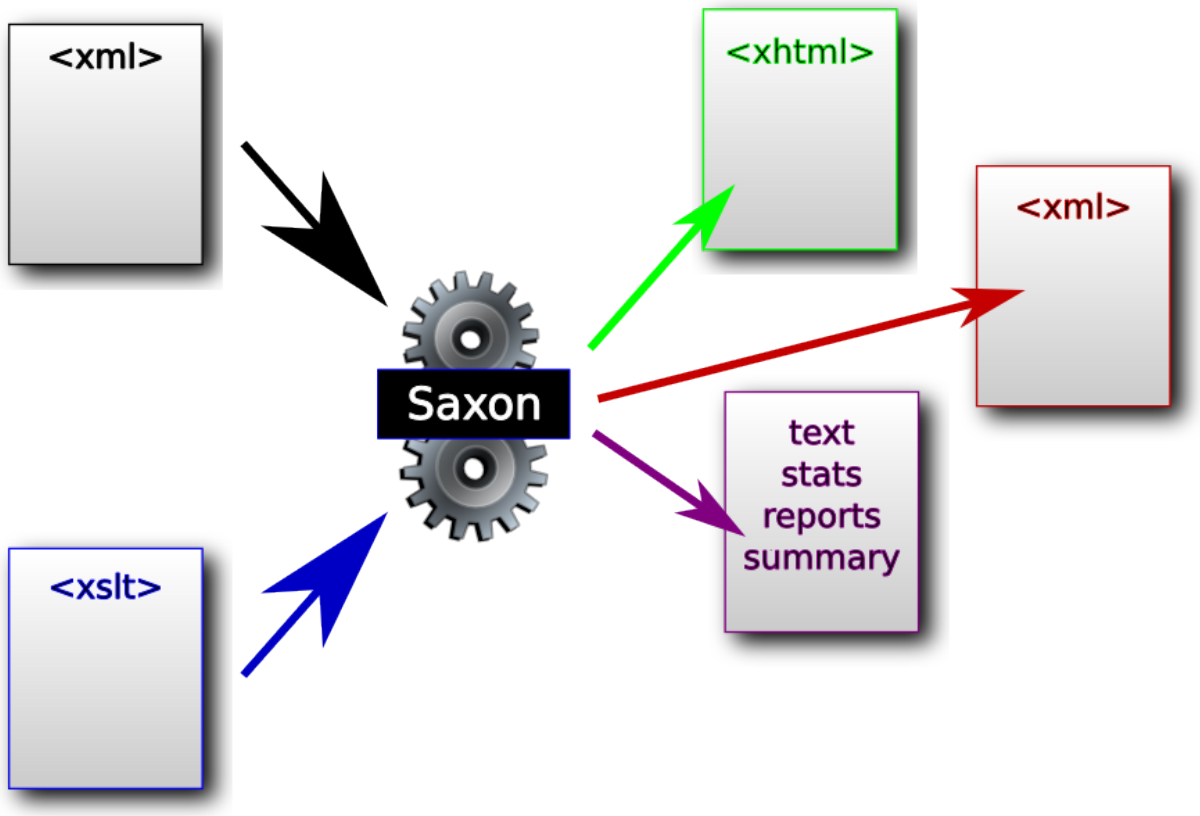


XSLT in the publishing workflow

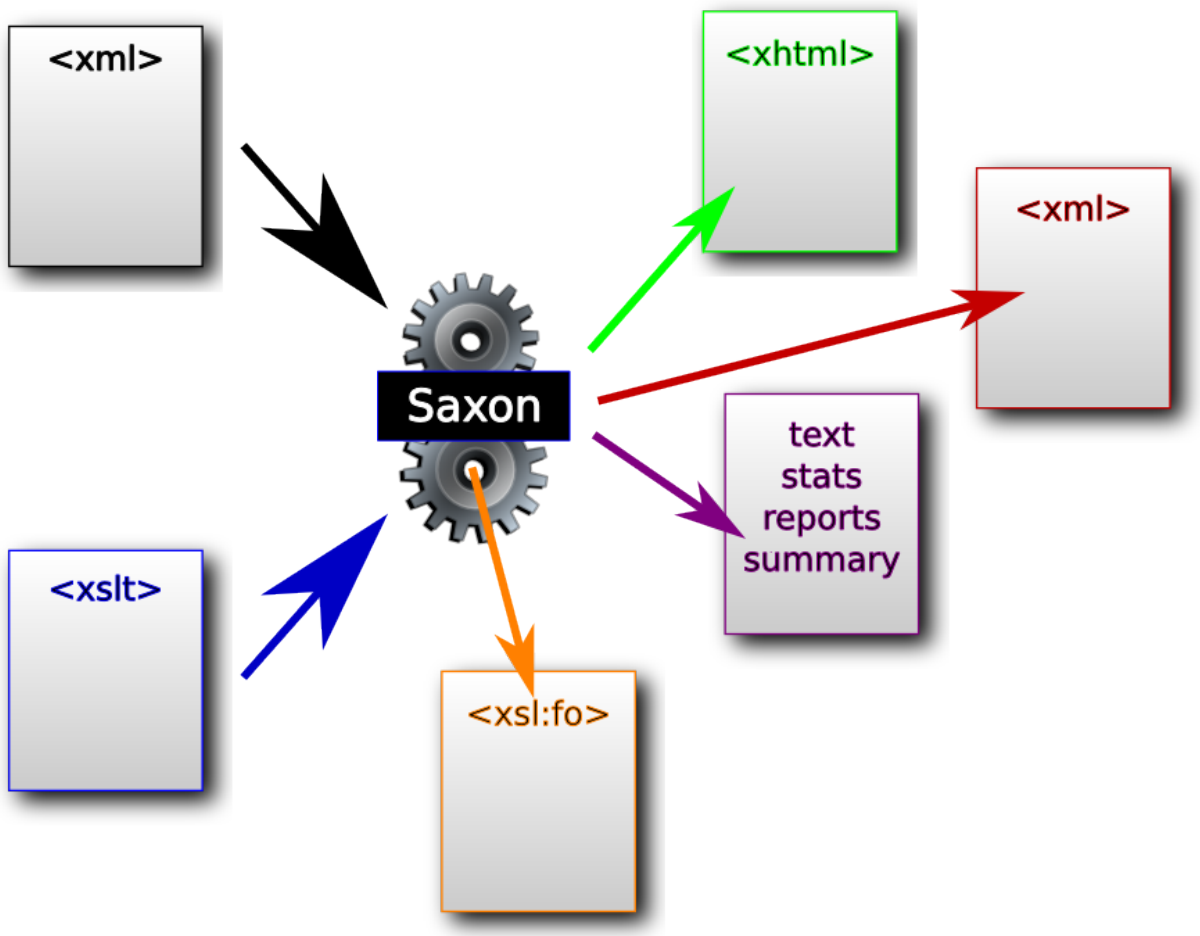


Try to elicit as many different XML output flavours as might be common, including TEI (of course), DocBook, SVG, Dublin Core or RDF (metadata), MARC, MODS etc. etc.

XSLT in the publishing workflow

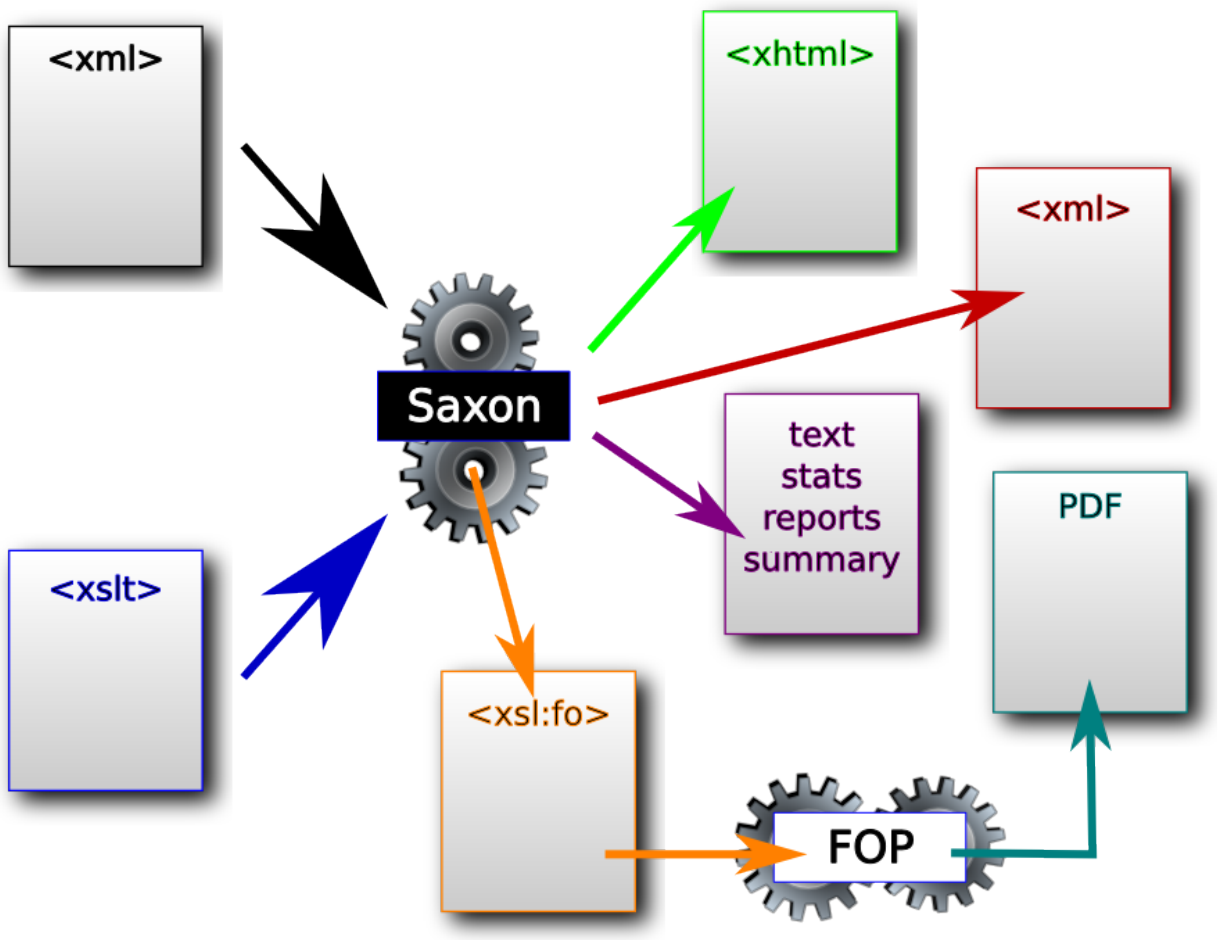


XSLT in the publishing workflow



Elicit what xsl:fo is for, and how it's used.

XSLT in the publishing workflow



An important point to make at this stage is that when we transform our typically rich TEI code into something else, we frequently throw away lots of information. For example, if we create a PDF for printing, we may throw away all the links, because they can't be followed from a piece of paper. Similarly, when we create XHTML, we may hide all sorts of key information

(such as components of the teiHeader) because they're too detailed to be effective components of a web page.

Some examples

- The Scandinavian Canadian Studies Journal scancan.net
- The Map of Early Modern London mapoflondon.uvic.ca
- The Colonial Despatches of Vancouver Island and British Columbia 1846-1871 bcgenesis.uvic.ca

What does XSLT look like?

To answer this question, we're going to create a simple XSLT file together.

You're welcome to type along with me in Oxygen if you'd like to.

We're going to start with the hamlet.xml example file, which you can find here:

http://web.uvic.ca/~mholmes/dhoxss2013/examples/hamlet.xml

Stress that it's absolutely not important that they understand everything that's happening here. The main idea is to get a sense of how transformations work, what templates look like, and what happens when you don't provide a template (default processing behaviour).

This is the process:

- Open the Hamlet file in Oxygen, and give a brief overview of its structure.
- Create a new XSLT file in Oxygen. Choose Customize, and select 2.0 and make sure the documentation elements are added.
- Delete the documentation elements for simplicity's sake (but explain this is BAD).
- Save the stylesheet as example_01.xsl in the examples folder.
- Draw attention to the root element and the namespaces defined on it. Explain that we're going to deal with namespaces much more carefully later, but for the moment, elicit the two additional namespaces we need to add (XHTML and TEI).
- Add the XHTML namespace as the default: xmlns="http://www.w3.org/1999/xhtml".
- Add the TEI namespace as the XPath default: xpath-default-namespace="http://www.tei-c.org/ns/1.0"

- Add the xsl:output element with the xhtml method. Explain this.
- Create the first (root) template, and the shell of an XHTML document in it. Elicit all the components of the XHTML shell.
- Put xsl:apply-templates in the body, and then run it in the XSLT debugger.
- Get the class to tell you what happened.
- Move on to suppressing stuff we don't want to see. Create empty templates for the header and the front. Show how these work to suppress these elements.
- Add a div template, and show how that works.
- Add a head template, and show how it produces h2.
- Add the following templates:
    - sp (speech): div (with a linebreak before it)
    - speaker: strong
    - l (line): just a linebreak after it
    - stage: em for italics, and precede and follow with a square bracket.

Again, the purpose here is not to understand exactly how all of this works; it's to demonstrate that even with a very simple XSLT transformation, you can start to get quite impressive results.

If all the students have followed along, you might give them ten or fifteen minutes to continue extending the stylesheet to see how they get on.