

# XSLT

The nuts and bolts of transforming XML

# Programming XSLT

- Unlike CSS, XSLT is really a fully fledged programming language.
- It's a *functional* (though some have argued just *declarative*) language, not a *procedural* or *object oriented* one.

# The upshot

- XSLT can do just about anything. And if it can't, you can write an extension to make it do what you want.
- If you've done programming in PHP, Java, C, BASIC, etc., it will take you a while to get XSLT. This is good news for the non-programmers: you're on a level playing field.
- You can use most of XSLT without fully understanding it.

# XSLT and XPath

- XPath is a way of addressing any node in an XML document.
- It's used *heavily* in XSLT: any time there's a match or a selection, it's done with XPath
- Like, XSLT, there's probably more than you'll use right away.

# XPath Basics

- XPath works in *steps*, which are combined into *paths*.
- Steps can be element names, attribute names (@+name), text(), comment(), and processing-instruction().
- Note what's missing: XML Declaration, DOCTYPE declaration, entities, CDATA. The former two are out-of-bounds, the latter have been resolved already by the time the XSL processor gets the document.

# Location Paths

- Steps can be combined into paths using the “/” character.
- Paths can be absolute (starting with “/”) or relative.
- XPath supports 3 “wildcards,” \*, node(), and @\*.
- You can also match multiple elements by separating them with “|”

# Shortcuts

- // means “all descendants”
- .. means the parent element
- . means the current element

# Predicates

- This is like the WHERE clause in an SQL statement.
- For example `//div[head = 'Chapter 1'` would get you the `<div>` with a `<head>Chapter 1</head>` inside it.
- The predicate contains an expression that evaluates to true or false.

# Axes

- So far, we've been looking at the compact syntax for XPath, there's a fuller syntax too, that allows for more control.
- This uses a system called *axes*.
- We've been using axes already, just not naming them. These are child, parent, self, attribute, and descendant-or-self.
- The full syntax uses the axis name, plus `::`, followed by the desired step.

# 8 more axes

- ancestor
- following-sibling
- preceding-sibling
- following
- preceding
- namespace
- descendant
- ancestor-or-self

# XPath functions

- You can do math in XPath expressions (+, -, \*, div, mod)
- There are also boolean, number, string, and node-set functions.
- So you can do things like get the number of characters contained in a <title> using `string-length(title)`.

# Using TEI

editorial markup

# TEI Projects

- The most frequent use for TEI is to produce marked up versions of existing texts.
- As we've seen already, texts are problematic.
- Marking up a text is an editorial act.

# Editing with TEI

- Any project using TEI will have to make several decisions:
  - What DTD?
  - What to tag? What not to tag?
  - Which hierarchy to use?

# Encoding Procedures

- 📌 Getting the text into digital form
  - 📌 OCR -> markup
  - 📌 Transcription/markup
  - 📌 Conversion
- 📌 Quality Control very important

# Encoding Practices

- 📌 Take a look at  
<http://www.diglib.org/standards/tei.htm>
- 📌 Five “levels” of encoding
- 📌 Standard values for attributes

# Assumptions

- TEI (and XML in general) lets you separate formatting information from structure and data
- ...but if you're marking up an existing text of historical value, visual features may be important.

# Assumptions


- ...are dangerous, but must be made anyway.
- Try to consider not only how you plan to use or display the texts, but also how others might use them.
- ...should be documented scrupulously.

# The TEI Header

- Provides you a way to document all of this stuff.
- What the document is, its context and classification, who was responsible for what, when, how it was done.

# Header Elements

```
<teiHeader>  
  <fileDesc>...</fileDesc>  
  <encodingDesc>...</encodingDesc>  
  <profileDesc>...</profileDesc>  
  <revisionDesc>...</revisionDesc>  
</teiHeader>
```

 <http://www.tei-c.org/P4X/HD.html>

# Extending TEI

- 📌 Don't do it unless you're certain the standard doesn't provide for what you want to do.
- 📌 Some extensions are standard themselves, like adding a `url` or `uri` attribute to `<xref>` and `<xptr>` (P5 reflects some of these practices).

# Managing Complexity

- 📌 Texts, and the things people want to do with them, are complicated.
- 📌 Keep things as simple as possible, but not simpler.
- 📌 Use existing standards and examples whenever possible.
- 📌 Don't be afraid to ask for help.