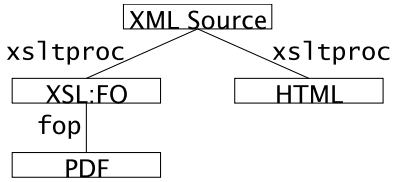
Crash Course in XSLT

Overview of XML and XSLT
Jan. 15, 2007
David Z. Maze

What is XSLT?

* "XML Stylesheet Language: Transformations"
* Convert XML to XML
  * ...to XHTML
  * ...to XSL:FO (...to PDF)
  * ...to other XML
* Convert XML to HTML
* Convert XML to text
* Convert XML to LaTeX, etc. (with care!)
* Language has XML syntax
* Accessible via C, Java, C#, standalone interpreters, dedicated hardware

Use Case: Documents



Use Case: XML to XML

* Application server expects SOAP-formatted message
* Change SOAP headers
* Convert from a different SOAP (or non-SOAP) format

Class Structure

* Three sessions, 15, 17, 22 January at 8:00
* Today: XML, Namespaces, XPath
* Wednesday: XSLT
* Next week: Details
* `http://stuff.mit.edu/iap/xslt/`

Some XML Specs

* Extensible Markup Language (XML) 1.0
  `http://www.w3.org/TR/xml`
* Namespaces in XML 1.0
  `http://www.w3.org/TR/xml-names`
* XML Path Language (XPath) 1.0
  `http://www.w3.org/TR/xpath`
* Extensible Stylesheet Language: Transformations (XSLT) 1.0
  `http://www.w3.org/TR/xslt`

What's in XML?

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:text>Hello, world!</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```


What's in XML? Elements

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:text>Hello, world!</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

* **Elements** are arranged in a tree structure
* Exactly one *document element*
* Every element has a *parent*
* Elements can have *children* that are text or other elements
* Tags are paired `<tag>...</tag>`
* Empty element shorthand `<tag/>`

What's in XML? Attributes

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:text>Hello, world!</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

* **Attributes** are "properties" of elements
* Meaning is typically specific to containing element
* Value always quoted, either single or double quotes
* Only has a text value
* Has a *parent* element

What's in XML? Text

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:text>Hello, world!</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

* **Text** can appear inside elements
* *Mixed content*: both text and elements appear inside the same element (e.g., XHTML's <p> tag)
* Text nodes always have a *parent* element
* *Entities* let you write "forbidden" characters: `&lt;` for <, `&gt;` for >, `&amp;` for &, `&apos;` for ', `&quot;` for "

XML Namespaces

* Every element and attribute has a *qualified name*
* (XPath spec says "expanded name" to same effect)
* Three parts: namespace prefix, **namespace URI, local name**

Namespace Bindings

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

* Element's local name is `stylesheet`, prefix is `xsl`, namespace URI is XSLT's
* Namespace-qualified names `pfx:name` must have a *namespace binding* `xmlns:pfx="urn:some:namespace"` on the current element or an ancestor

Default Namespaces

```
<stylesheet version="1.0"
 xmlns="http://www.w3.org/1999/XSL/Transform">
```

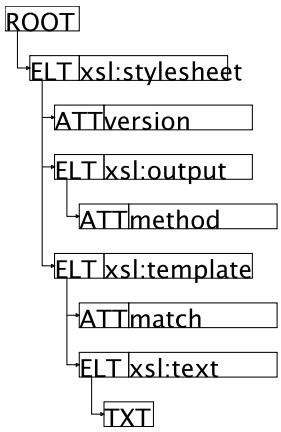
* Element's local name is `stylesheet`, prefix is none, namespace URI is XSLT's
* Attribute's local name is `version`, prefix is none, namespace URI is none
* *Default namespace doesn't apply to attributes*

Namespace Bindings are Inherited

```
<root xsl:version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:stylesheet version="1.0">
```

* Root element's local name is `root`, prefix is none, namespace URI is none
* `xsl:version` attribute has XSLT namespace
* Child element's local name is `stylesheet`, prefix is `xsl`, namespace URI is XSLT's
* `version` attribute has null namespace
* *Can't have two attributes on the same element with same expanded name*

XPath Data Model

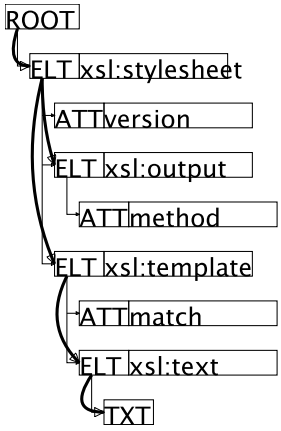


* Nodes have a *type*, *name*, *string value*
* Fundamental data type is *node set*

XPath Syntax

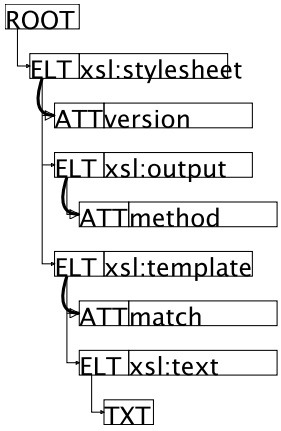
* A path has a series of *steps*
* *`start/step/step/...`*
* Each step has an *axis* and a *name* (or `*` or `pfx:*`)
* *`axis`*`::`*`pfx:lcl`*`[`*`pred`*`]`
* `/child::xsl:stylesheet/child::xsl:template`
* `/descendant::*[attribute::id]`
* Names always match by *expanded name*; namespace prefixes in expression and document need to be bound to the same namespace
* Trivial axis: *self*

XPath Axes: Child



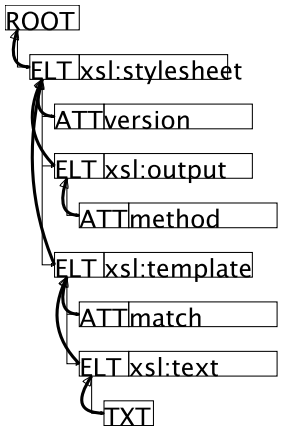
* Child step goes from element or root node to its element or text children
* Shorthand: `name` with no axis specifier means `child::name`

XPath Axes: Attribute



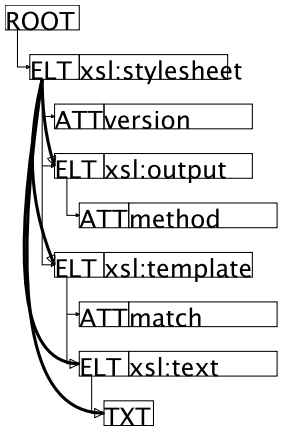
* Attribute step goes from element node to its attribute children
* Shorthand: `@name` means `attribute::name`

XPath Axes: Parent



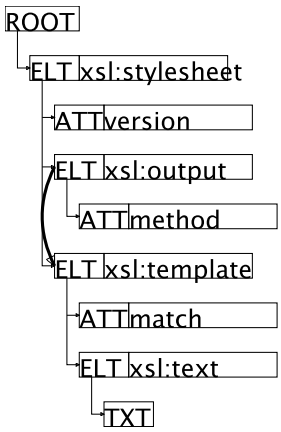
* Parent step goes from any node to its immediate element or root parent

XPath Axes: Descendant (Ancestor)



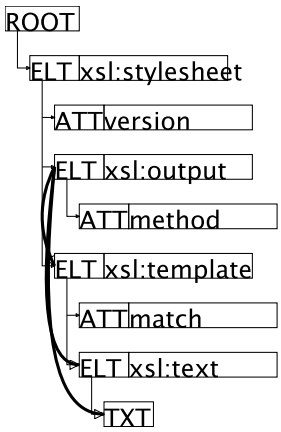
* Descendant step goes from element or root node to its element or text children, and their children, and...
* Descendant-or-self step also includes current node
* Ancestor step goes from any node to its parent, and its parent, and...
* Ancestor-or-self step also includes current node
* Shorthand: `foo//bar` is really `foo/descendant-or-self::*/child::bar`

XPath Axes: Following (Preceding) Sibling



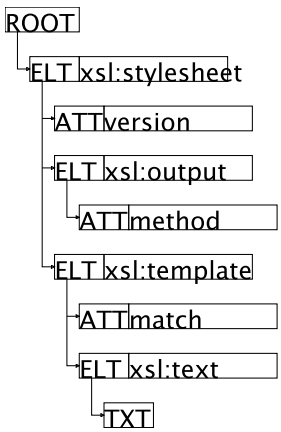
* Following sibling step goes from element or text node to following element or text nodes with the same parent
* Preceding sibling is similar

XPath Axes: Following (Preceding)



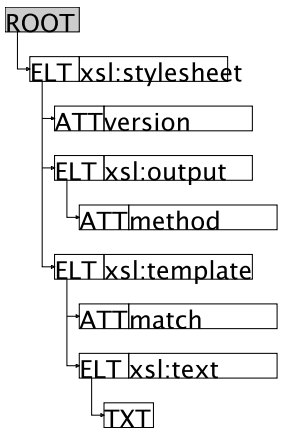
* Following step goes from element or text node to following element or text nodes anywhere in the document
* Preceding is similar
* In practice, probably the most expensive of the axes

XPath Evaluation



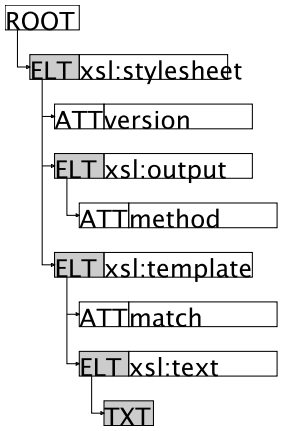
* `//@match`
* `/descendant-or-self::*/attribute::match`

XPath Evaluation



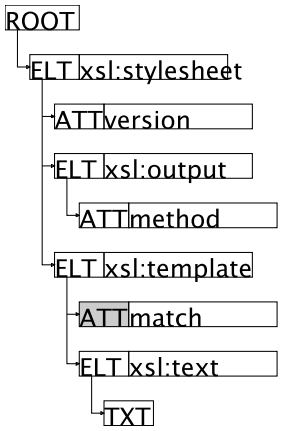
* `//@match`
* `/descendant-or-self::*/attribute::match`

XPath Evaluation



* `//@match`
* `/descendant-or-self::*/attribute::match`

XPath Evaluation

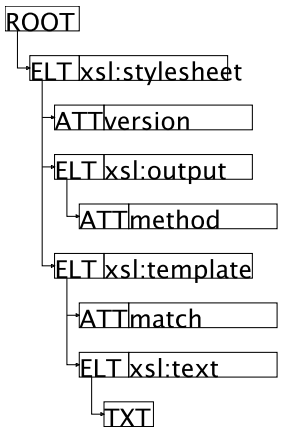


* `//@match`
* `/descendant-or-self::*/attribute::match`

XPath Predicates

* Only consider nodes where some expression is true
* Wider expression language here
* *Node sets* are considered "true" if non-empty
* *Strings* are considered "true" if non-empty

XPath Predicates



* `//*[@match]`
* Each descendant node is considered the *context* node for evaluating the attribute step
* "Set of element nodes anywhere in the document such that, for each node, there is at least one attribute named `match`"

XPath Union

* `a|*[@foo]`
* "All a's, plus all children that have a foo attribute"
* If any node is in both sets, it only appears in the result once

More XPath Expressions

* `count()`: size of parameter node set
* `local-name()`, `namespace-uri()`
* `position()`, `last()` within current node set
* Equality, comparison (`a < 3`)
* Numeric operations(`+ - * div`)
* `$variable`

More Little Details

* Predicate `...[1]` means `...[position() = 1]`
* Similarly `child::foo[last()]`
* Paths can begin with `/`, `$var`, or nothing

Caution the first

```
<a>
  <b>1</b>
  <b>2</b>
</a>
```

* `/a` matches the root element
* `/a[b = 1]` does too
* `/a[b != 1]` does too!
* `/a[not(b = 1)]` doesn't

Caution the second

```
<a>
  <b>
    <c/>
  </b>
  <b>
    <c/>
  </b>
</a>
```

* `//c` matches both c elements
* e.g. `/a/b[1]` matches only the first b
* `//c[1]` matches both cs!
* `/descendant-or-self::*/child::c[position() = 1]`
* Consider `(//c)[1]`

What Haven't I Said?

* XML character entities, e.g. ` `
* XML processing instructions
* XML DTDs
* XPath `namespace` axis

Next Time

* So far: XPath in isolation
* Next time: XSLT!