

# XSLT 1.0

Master Recherche — Données semi-structurées

Pierre Senellart  
pierre@senellart.com



9th January 2007

# Motivation

- XML is used in a large number of applications, either **data-centric**, or **document-centric**. In either case, there is a **need** for **transforming** documents from one XML formalism to another, or from one XML formalism to another kind of formalism (HTML, text-based formats. . . ).
- Can be achieved with regular libraries for parsing XML documents (DOM, SAX) in traditional programming languages, but this tends to be a bit **tedious**.
- A **declarative** approach would be welcome (more readable, more concise, more adapted to a public of non-programmers).

# XSLT

- **XSL** (= e**X**tensible **S**tylesheet **L**anguage) was originally planned as a language for the formatting of XML documents. Has been split between a presentation format for printed text (XSL-FO), and a transformation language for XML (XSLT), which is used now in a much broader context.
- **Declarative, side-effect-free** language, for transforming XML trees into text, HTML or XML.
- Relies heavily on the **XPath** expression language for **selecting** nodes in the XML tree.
- One limitation of this approach: the vast majority of implementations require to store the original document into memory, which makes it impossible to use it with very large (>100MB) documents.

# Outline

1 Introduction

2 Preliminaries

- XML
- Namespaces
- Reference Information

3 XPath 1.0

4 XSLT 1.0

5 Beyond XSLT 1.0

# Terminology

```
<a toto="3">  
  <b titi='tutu'><c /></b>  
  <d>tata</d>  
</a>
```

<a ...>, <b>, <d>

</a>, </b>, </d>

<c />

a, b, c, d

toto, titi

3, tutu

tata

opening tags

closing tags

empty-element tag

element names

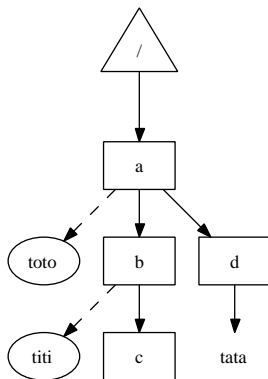
attribute name

attribute value

character data or text content

# Tree Data Model for XML

```
<a toto="3">  
  <b titi='tutu'><c /></b>  
  <d>tata</d>  
</a>
```



## Node kinds



(Text nodes containing only whitespace are not shown)

# Motivation for Namespaces

Each **XML application** comes with its own **vocabulary** (set of element and attribute names).

## Problem

How to mix elements from different vocabularies in the same document, when they may share the same element names? Needed for instance when describing a **transformation** from one vocabulary to another.

## Solution

Assign to each vocabulary, a **namespace**, which is a **formal string** (an URI), supposed to be globally unique. In an XML document, the namespace URI will be **bound** to a prefix that represents this URI more concisely.

## Examples of Namespaces

Application	Namespace URI	Usual prefix
HTML	<code>http://www.w3.org/1999/xhtml</code>	<i>none</i>
XML	<code>http://www.w3.org/XML/1998/namespace</code>	<code>xml</code> <sup>1</sup>
XSLT	<code>http://www.w3.org/1999/XSL/Transform</code>	<code>xsl</code>
XML Schema	<code>http://www.w3.org/2001/XMLSchema</code>	<code>xs, xsd</code>
XML Catalogs	<code>urn:oasis:names:tc:entity:xmlns:xml:catalog</code>	<i>none</i>
TMX	<code>http://www.lisa.org/tmx14</code>	<i>none</i>

### Remark

*The namespace URIs are only **formal**: It may not correspond to the URL of a real document, and no Web access is performed by the XML parser. URIs are only used because it is easy to produce a globally unique URL, in a domain name under one's control.*

<sup>1</sup>The `xml` prefix does not need to be declared with an `xmlns:xml`, it is a default in all XML documents.

## Namespaces in XML

A prefix `pf` is bound to a given namespace URI `http://uri` by adding a **pseudo-attribute** `xmlns:pf` to an XML element. Then, all elements that are descendants of this element (and this element itself) are bound to the namespace `http://uri` if their name starts with “`pf:`”. Element names without a prefix can also be bound to a namespace, with a `xmlns` pseudo-attribute. Otherwise, elements are bound to the so-called **default namespace**.

```
<toto xmlns:pf="http://uri">
  <pf:titi> <!-- This one is in the ns -->
    <titi>tutu</titi>           <!-- This one is not -->
  </pf:titi>
  <tutu xmlns="http://uri"> <!-- But this one is -->
    <toto />                 <!-- And this one also -->
  </tutu>
</toto>
```

# References

- <http://www.w3.org/TR/xml/>
- <http://www.w3.org/TR/xml-infoset/>
- <http://www.w3.org/TR/xml-names/>
- *XML in a nutshell*, Eliotte Rusty Harold & W. Scott Means, O'Reilly

# Outline

- 1 Introduction
- 2 Preliminaries
- 3 XPath 1.0**
  - Basics
  - Path Expressions
  - Axes
  - Operators and Functions
  - Reference Information
  - Exercise
- 4 XSLT 1.0
- 5 Beyond XSLT 1.0

# XPath

- An **expression language** to be used in another host language (e.g., XSLT).
- Allows the description of **paths** in an XML tree, and the retrieving of nodes that match these paths.

## Example

`//*[ @msg="Hello world" ]` is an XPath expression which retrieves all elements with a `msg` attribute set to "Hello world".

# XPath 1.0 Type System

Four primitive types:

Type	Description	Literals	Examples
boolean	Boolean values	<i>none</i>	<code>true()</code> , <code>not(\$a=3)</code>
number	Floating-point numbers	12, 12.5	<code>1 div 33</code>
string	Character strings	"to", 'ti'	<code>concat('Hello','!')</code>
nodeset	Unordered sets of nodes	<i>none</i>	<code>/a/b[c=1 or @e]/d</code>

The `boolean()`, `number()`, `string()` functions **convert** types into each other (no conversion to nodesets is defined), but this conversion is done in an **implicit** way most of the time.

Rules for **converting to a boolean**:

- A number is true if it is neither 0 nor *NaN*.
- A string is true if its length is not 0.
- A nodeset is true if it is not empty.

## Rules for **converting a nodeset to a string**:

- The string value of a nodeset is the string value of its first item in document order.
- The string value of an element or document node is the concatenation of the character data in all text nodes below.
- The string value of a text node is its character data.
- The string value of an attribute node is the attribute value.

### Examples (Whitespace-only text nodes removed)

```
<a toto="3">  
  <b titi='tutu'><c /></b>  
  <d>tata</d>  
</a>
```

<code>string(/)</code>	<code>"tata"</code>
<code>string(/a/@toto)</code>	<code>"3"</code>
<code>boolean(/a/b)</code>	<code>true()</code>
<code>boolean(/a/e)</code>	<code>false()</code>

# XPath Context

All XPath expressions are evaluated in a specific context, which includes:

a **context node** which is the **current node** in the tree; it is referred to in XPath by the symbol “.”.

a **context length** which is a positive integer, indicating the **size** of a contextual list of nodes; it can be known by using the function **last()**.

a **context position** which is a positive integer lesser or equal to the context length, indicating the **position** into the contextual list of nodes; it can be known by using the function **position()**.

## Path Expressions

A path expression returns a nodeset and has one of the following forms:

`/step1/step2/.../stepn` **absolute** path expression

`step1/step2/.../stepn` path expression **relative** to the **context node**

There may be any number of steps, each of these has the following form:

`axis::node-test[P1][P2]...[Pn]`

`axis` is an **axis name** (cf. slide 18), indicating what the direction of the step in the tree is. The `axis::` part may be omitted, the **child** axis is the default.

`node-test` is a **node test** (cf. following slide), indicating the kind of nodes to select.

`Pi` is a **predicate**, that is, any XPath expression, evaluated as a boolean, indicating an additional condition. There may be no predicates at all.

`.` is a special step, which refers to the current node. Thus, `./toto` means the same thing as `toto`.

## Node Tests

A node test has one of the following forms:

`node()` any node.

`text()` any text node.

\* any element (or any attribute for the attribute axis).

`ns:*` any element or attribute in the namespace bound to the prefix `ns`.

`ns:toto` any element or attribute whose name is `ns:toto`

### Examples

`a/node()` selects all nodes which are children of a node, itself child of the context node.

`xsl:*` selects all elements whose namespace is `ns` and that are children of the context node.

`/*` selects the top-level element node.

# Axes

`child` (default axis).

`parent` Parent node.

`attribute` Attribute nodes.

`descendant` Descendants, excluding the node itself.

`descendant-or-self` Descendants, including the node itself.

`ancestor` Ancestors, excluding the node itself.

`ancestor-or-self` Ancestors, including the node itself.

`following` Following nodes in `document order` (except descendants).

`following-sibling` Following siblings in `document order`.

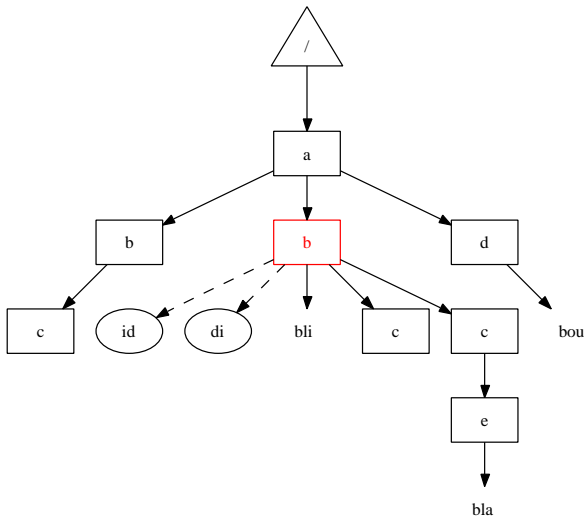
`preceding` Preceding nodes in `document order` (except ancestors).

`preceding-sibling` Preceding siblings in `document order`.

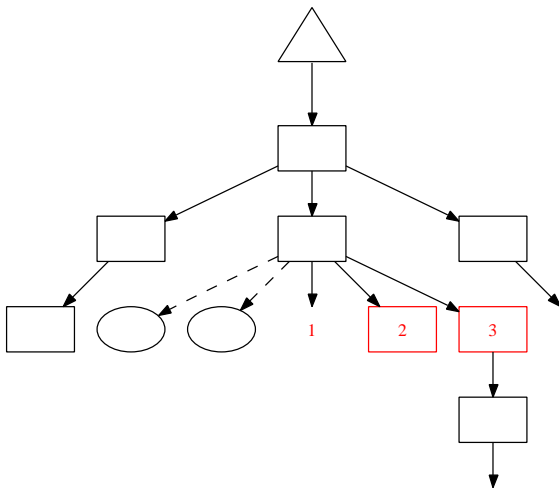
## Notes about Axes

- An empty nodeset is always returned for parent, attribute, ancestor, ancestor-or-self, following-sibling, preceding, preceding-sibling when the context node is a **document node**.
- An empty nodeset is always returned for child, attribute, descendant, descendant-or-self, following-sibling, preceding-sibling when the context node is an **attribute node**.
- An empty nodeset is always returned for child, attribute, descendant, descendant-or-self when the context node is a **text node**.
- An attribute node has a parent (the element on which it is located), but an attribute node is not one of the children of its parent.
- In a predicate following a step, the context size is initialized to the number of nodes matched; the predicate is evaluated for each different node matched with a different context position. For **forward** axes, positions follow the document order; for **backward** axes, they are in reverse order.

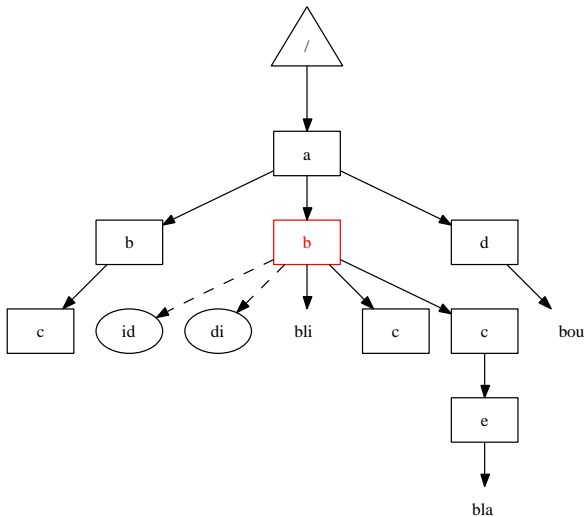
## child Axis



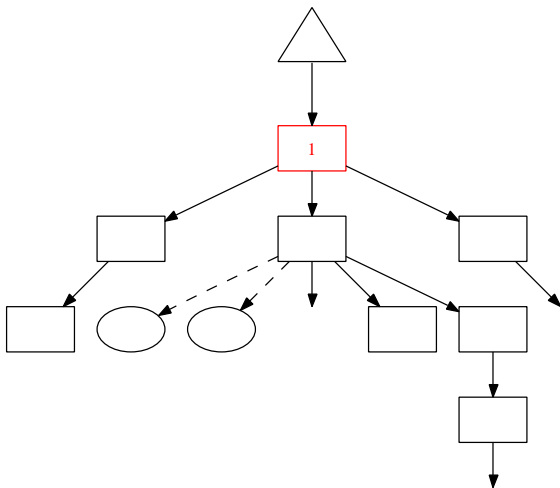
## child Axis



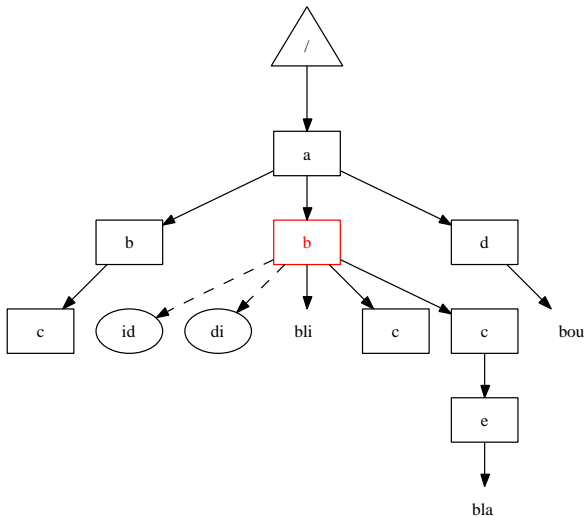
# parent Axis



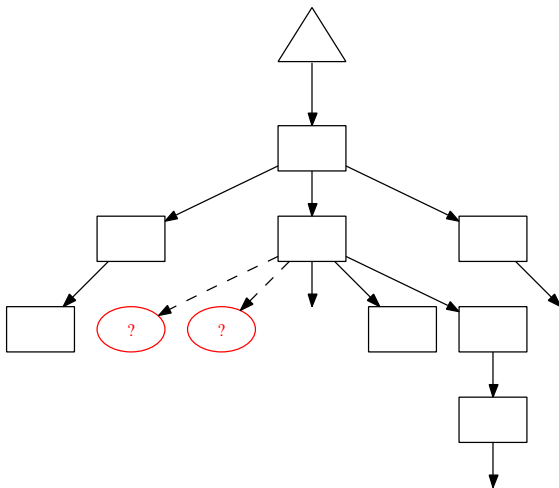
# parent Axis



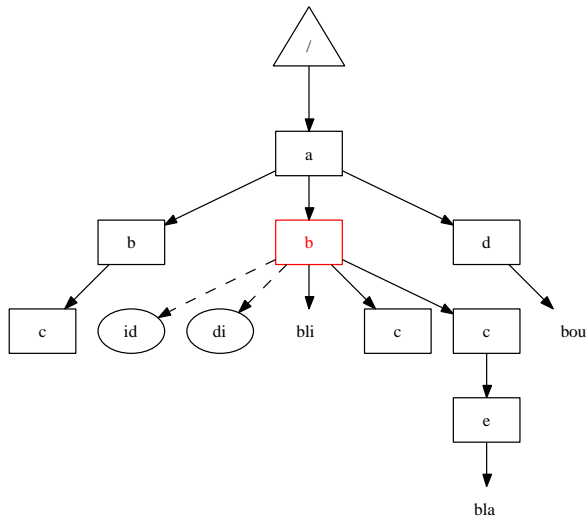
## attribute Axis



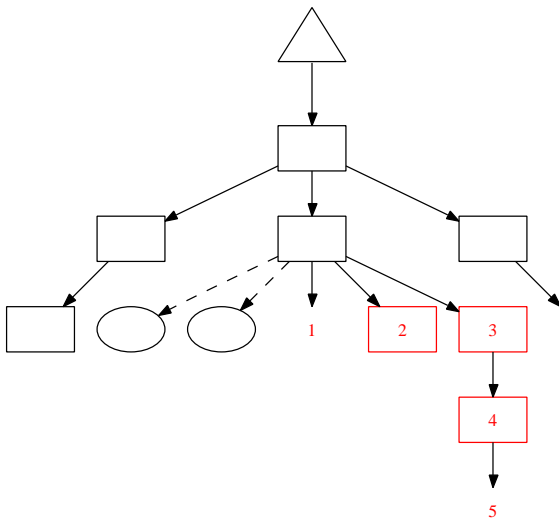
## attribute Axis



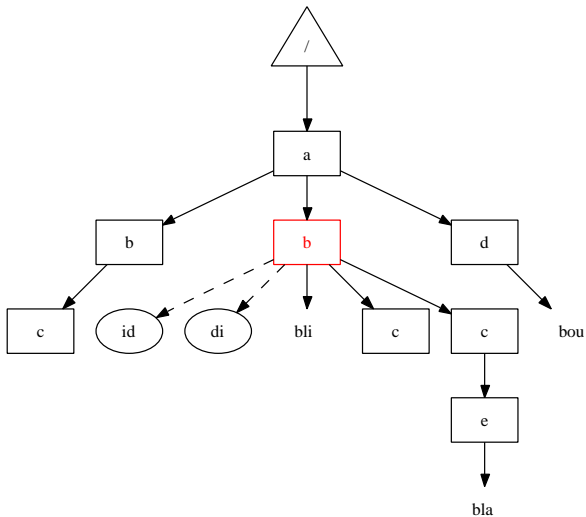
# descendant Axis



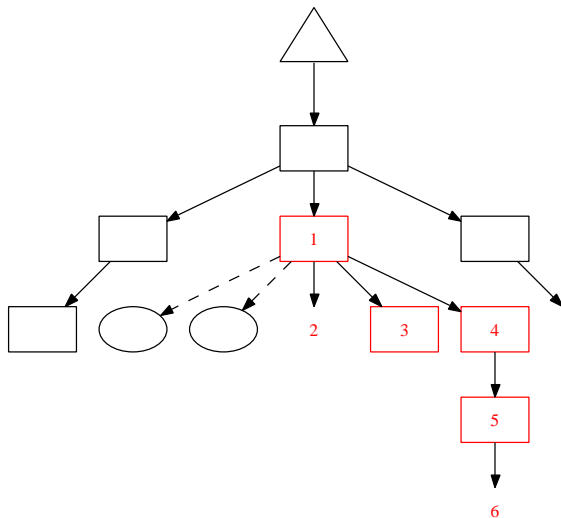
# descendant Axis



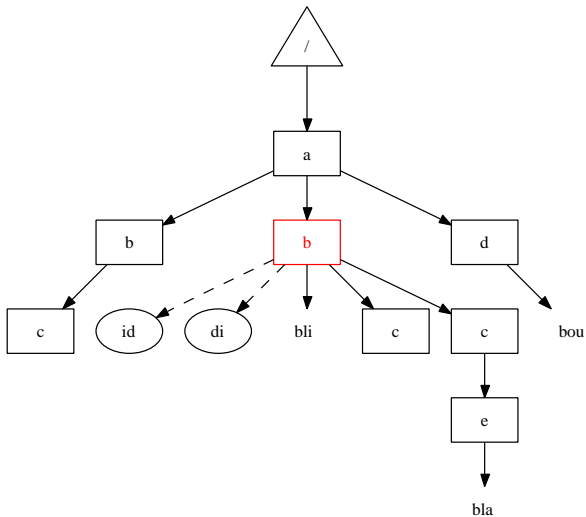
## descendant-or-self Axis



# descendant-or-self Axis

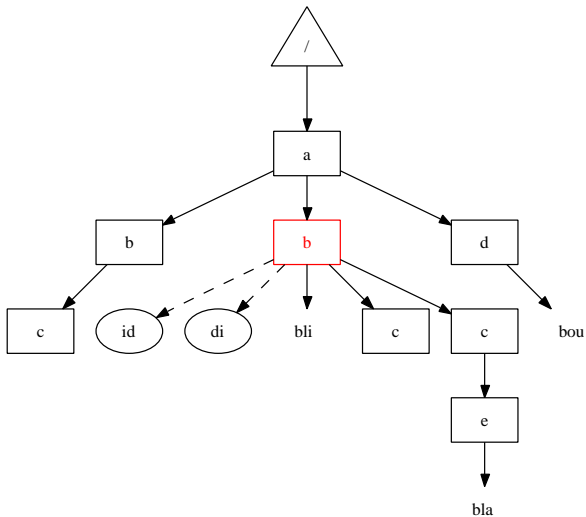


## ancestor Axis

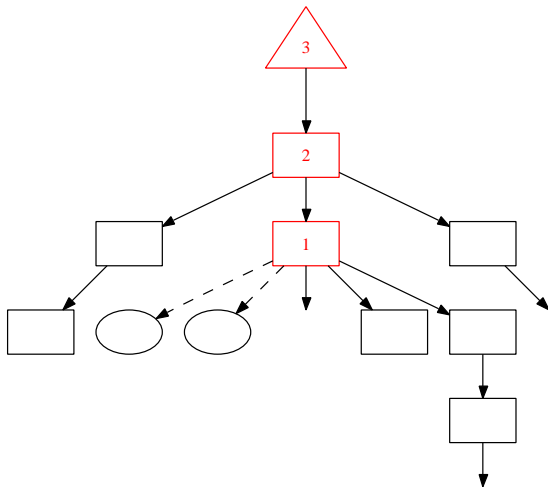




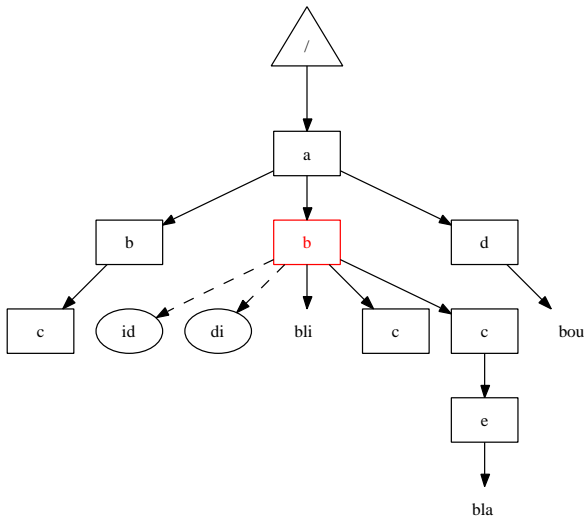
## ancestor-or-self Axis



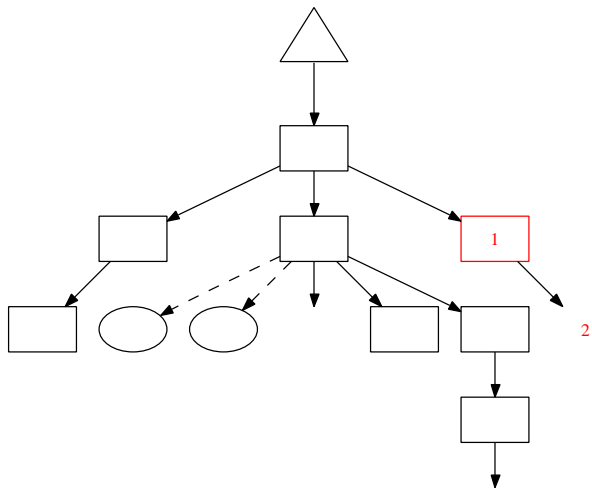
## ancestor-or-self Axis



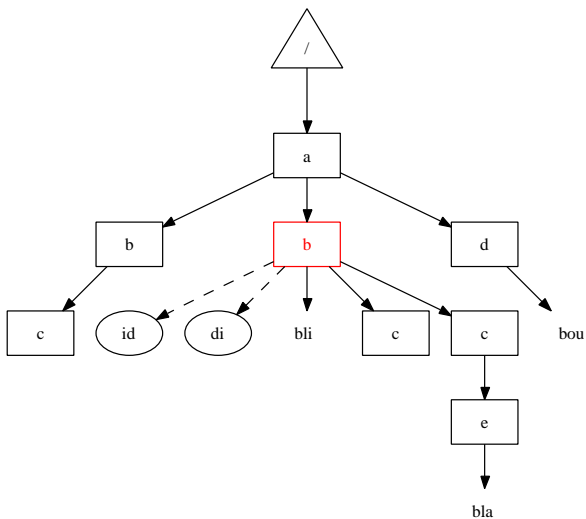
## following Axis



## following Axis

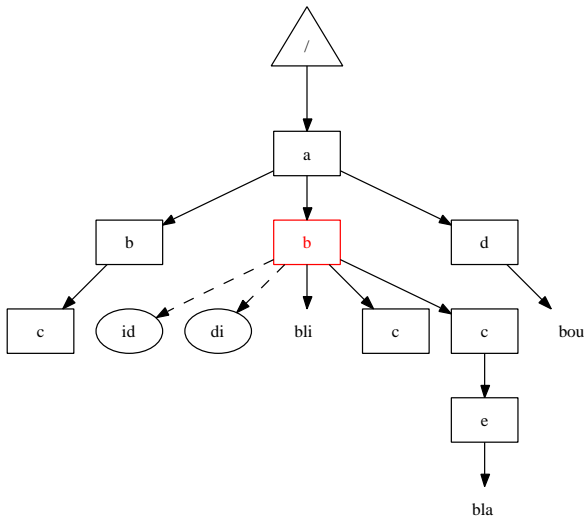


## following-sibling Axis

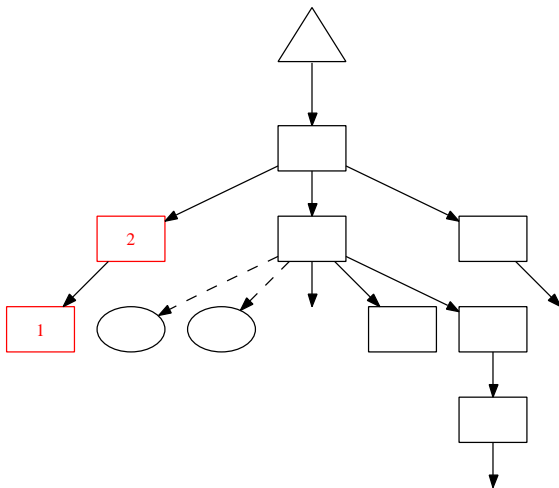




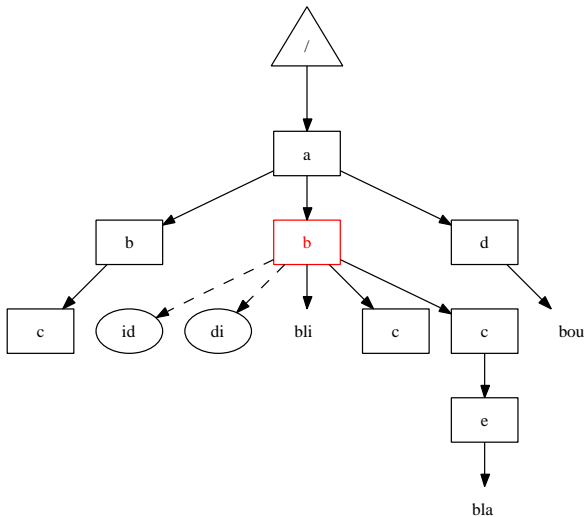
## preceding Axis



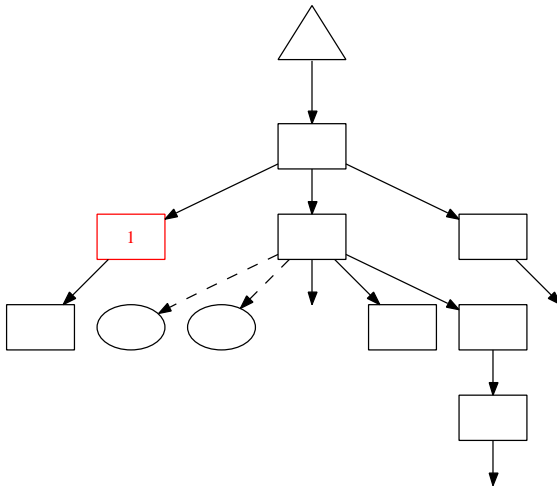
# preceding Axis



# preceding-sibling Axis



# preceding-sibling Axis



## Abbreviations

<code>toto</code>	<code>child::toto</code>
<code>..</code>	<code>parent::node()</code>
<code>@toto</code>	<code>attribute::toto</code>
<code>a//b</code>	<code>a/descendant-or-self::node()/b</code>
<code>//a</code>	<code>/descendant-or-self::node()/a</code>
<code>a[5]</code>	<code>a[position()=5]</code>

### Examples

`@b` selects the `b` attribute of the context node.

`../*` selects all siblings of the context node, itself included (unless it is an attribute node).

`//@blah` selects all `blah` attributes wherever their position in the document.

`titi[2]/tutu` selects the `tutu` children of the second `titi` child of the context node.

## Operators

The following operators can be used in XPath.

`+`, `-`, `*`, `div`, `mod` standard arithmetic operators (Example: `1+2*-3`).

**Warning!** `div` is used instead of the usual `/`.

`or`, `and` boolean operators (Example: `@a and c=3`)

`=`, `!=` equality operators. Can be used for strings, booleans or numbers. **Warning!** `//a!=3` means: there is an `a` element in the document whose string value is different from 3.

`<`, `<=`, `>=`, `>` relational operators (Example: `($a<2) and ($a>0)`).

**Warning!** Can only be used to compare numbers, not strings. If an XPath expression is embedded in an XML document (this will be the case with XSLT), `<` must be escaped as `&lt;`.

`|` union of nodesets (Example: `node()|@*`)

### Remark

`$a` is a *reference* to the variable `a`. Variables can not be defined in XPath, they can only be referred to.

## Node Functions

- `count($s)` returns the **number of items** in the nodeset `$s`
- `local-name($s)` returns the **name** of the first item of the nodeset `$s` in document order, **without** the namespace prefix; if `$s` is omitted, it is taken to be the context item
- `namespace-uri($s)` returns the **namespace URI** bound to the prefix of the name of the first item of the nodeset `$s` in document order; if `$s` is omitted, it is taken to be the context item
- `name($s)` returns the **name** of the first item of the nodeset `$s` in document order, **including** its namespace prefix; if `$s` is omitted, it is taken to be the context item

## String Functions

`concat($s1, ..., $sn)` concatenates the strings `$s1, ..., $sn`

`starts-with($a,$b)` returns `true()` if the string `$a` starts with the string `$b`

`contains($a,$b)` returns `true()` if the string `$a` contains the string `$b`

`substring-before($a,$b)` returns the substring of `$a` before the first occurrence of `$b`

`substring-after($a,$b)` returns the substring of `$a` after the first occurrence of `$b`

`substring($a,$n,$l)` returns the substring of `$a` of length `$l` starting at index `$n` (indexes start from 1). `$l` may be omitted.

`string-length($a)` returns the length of the string `$a`

`normalize-space($a)` removes all leading and trailing whitespace from `$a`, and collapse all whitespace sequence to a single character

`translate($a,$b,$c)` returns the string `$a`, where all occurrences of a character from `$b` has been replaced by the character at the same place in `$c`.

## Boolean and Number Functions

`not($b)` returns the **logical negation** of the boolean `$b`

`sum($s)` returns the **sum** of the values of the nodes in the nodeset `$s`

`floor($n)` rounds the number `$n` to the **next lowest** integer

`ceiling($n)` rounds the number `$n` to the **next greatest** integer

`round($n)` rounds the number `$n` to the **closest** integer

### Examples

`count(//*)` returns the number of elements in the document

`normalize-space(' titi toto ')` returns the string "titi toto"

`translate('baba','abcdef','ABCDEF')` returns the string "BABA"

`round(3.457)` returns the number 3

## XPath 1.0 Implementations

Large number of implementations, cf. also all the XSLT implementations of slide 66.

`libxml2` Free **C** library for parsing XML documents, supporting XPath.

`java.xml.xpath` **Java** package, included with JDK versions starting from 1.5.

`System.Xml.XPath` **.NET** classes for XPath.

`XML::XPath` Free **Perl** module, includes a command-line tool.

`DOMXPath` **PHP** class for XPath, included in PHP5.

`PyXML` Free **Python** library for parsing XML documents, supporting XPath.

# References

- <http://www.w3.org/TR/xpath>
- *XML in a nutshell*, Eliotte Rusty Harold & W. Scott Means, O'Reilly

## Exercise

```
<a>
  <b><c /></b>
  <b id="3" di="7">bli <c /><c><e>bla</e></c></b>
  <d>bou</d>
</a>
```

We suppose that all text nodes containing only whitespace are removed from the tree.

- Give the result of the following XPath expressions:
  - ▶ `//e/preceding::text()`
  - ▶ `count(//c|//b/node())`
- Give an XPath expression for the following problems, and the corresponding result:
  - ▶ Sum of all attribute values
  - ▶ Text content of the document, where every “b” is replaced by a “c”
  - ▶ Name of the child of the last “c” element in the tree

# Outline

- 1 Introduction
- 2 Preliminaries
- 3 XPath 1.0
- 4 XSLT 1.0**
  - Basics
  - Declarations
  - Template Rules
  - Additional Functions
  - Reference Information
- 5 Beyond XSLT 1.0

# A Hello World! Stylesheet

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="xml" encoding="utf-8" />

<xsl:template match="/">
  <hello>world</hello>
</xsl:template>
</xsl:stylesheet>
```

General structure of a stylesheet:

- A top-level `<xsl:stylesheet>` element, with a `version="1.0"` attribute.
- Some **declarations** (all elements except `<xsl:template>` ones), in this case just a `<xsl:output>` one, which specifies to generate an XML document, with the UTF-8 encoding (actually, both were defaults).
- Some **template rules**. First, the **current node** is the document node.

## Invocation of an XSLT Stylesheet

An XSLT stylesheet may be invoked:

- **Programmatically**, through one of the various XSLT library for the different programming languages. For the case of Web publishing, this would probably be by a CGI, PHP, ASP, JSP... script.
- Through a **command line** interface.
- In the case of the Web publishing, it can also be invoked on the **client side** through the XSLT engines integrated to most browsers. When a browser retrieves an XML document, a styling processing instruction may be included which tells the browser where to find an XSLT stylesheet for displaying it:

```
<?xml-stylesheet
  href="toto.xsl" type="application/xml" ?>
<doc>
  <titi />
</doc>
```

# Stylesheet Output

```
<xsl:output
  method="html"
  encoding="iso-8859-1"
  doctype-public=
    "-//W3C//DTD HTML 4.01//EN"
  doctype-system=
    "http://www.w3.org/TR/html4/strict.dtd"
  indent="yes" />
```

- `method` is either `xml` (default), `html` or `text`.
- `encoding` is the desired encoding of the result.
- `doctype-public` and `doctype-system` makes it possible to add a document type declaration in the resulting document.
- `indent` specifies whether the resulting XML document will be indented (default is `no`).

# Importing Stylesheets

```
<xsl:import href="toto.xsl" />
```

Template rules are imported this way from another stylesheet.

`<xsl:import>` must be the **first** declaration of the stylesheet.

## Global Variables and Parameters

```
<xsl:param name="nom" select="'John Doe'" />
<xsl:variable name="pi" select="3.14159" />
```

Global parameters are passed to the stylesheet through some **implementation-defined** way. The `select` attribute gives the default value, in case the parameter is not passed, as an XPath expression.

Global variables, as well as local variables which are defined in the same way inside template rules, are immutable in XSLT, since it is a side-effect-free language.

The `select` content may be replaced in both cases by the content of the `xsl:param` or `xsl:variable` elements.

## Handling Whitespace

```
<xsl:strip-space elements="*" />
<xsl:preserve-space elements="para poem" />
```

Both elements require a set of space-separated **node tests** as their `elements` attribute. `<xsl:strip-space>` specifies the set of nodes whose whitespace-only text child nodes will be removed, and `<xsl:preserve-space>` allows for exceptions to this list.

## The `<xsl:template>` Element

```
<xsl:template
  match="toto/titi/@tutu"
  priority="20"
  mode="premier">
  <xsl:text>The tutu is: </xsl:text>
  <xsl:value-of select="." />
  <xsl:copy-of select="*" />
</xsl:template>
```

The `select` attribute is a condition on the **current node** for this rule to be processed. It is an XPath expression which must match the current node, with for context node either the current node or one of its ancestor (`select="ancestor::*"` will not work for instance).

The `mode` is optional; if present, the **same mode** must be requested when using `<xsl:apply-templates>`; the `priority` manages **conflicts** between rules (cf. slide 60).

`<xsl:text>` creates a **text node**. Text nodes are also created when character data is written directly in the template body, but this will also put all surrounding whitespace in the text node.

`<xsl:value-of>` **evaluates** an XPath expression and add the result to the result tree.

`<xsl:copy-of>` is similar, it creates a **deep copy** of the node that results from the evaluation of the XPath expression of its `select` attribute.

## Applying a Template

```
<xsl:template match="toto">
  <ul><xsl:apply-templates select="titi" /></ul>
</xsl:template>
```

```
<xsl:template match="titi">
  <li><xsl:value-of select="." /></li>
</xsl:template>
```

```
<toto>
  <titi>23</titi>
  <titi>24</titi>
</toto>
```

→

```
<ul>
  <li>23</li>
  <li>24</li>
</ul>
```

`<xsl:apply-templates>` may also specify a mode through its `mode` attribute.

## Template Rule Conflicts

- Rules from imported stylesheets are **overridden** by rules of the stylesheet which imports them.
- Rules with **highest priority** prevail. If no priority is specified on a template rule, a default priority is assigned according to the **specificity** of the XPath expression (the more specific, the highest).
- If there are still conflicts, it is an error.
- If no rules apply for the node currently processed (the document node at the start, or the nodes selected by a `<xsl:apply-templates>` instruction), **built-in** rules are applied:

```
<xsl:template match="*|/">  
  <xsl:apply-templates select="node()" />  
</xsl:template>
```

```
<xsl:template match="@*|text()">  
  <xsl:value-of select="." />  
</xsl:template>
```

## Conditional Constructs

```
<xsl:choose>
  <xsl:when test="$year mod 4">no</xsl:when>
  <xsl:when test="$year mod 100">yes</xsl:when>
  <xsl:when test="$year mod 400">no</xsl:when>
  <xsl:otherwise>yes</xsl:otherwise>
</xsl:choose>

<xsl:value-of select="count(a)"/>
<xsl:text> item</xsl:text>
<xsl:if test="count(a)>1">s</xsl:if>
```

`<xsl:otherwise>` is optional. There can be any number of `<xsl:when>`, only the content of the first matching one will be processed.

## Named Templates

```
<xsl:template name="factorielle">
  <xsl:param name="n" />

  <xsl:choose>
    <xsl:when test="$n<=1">1</xsl:when>
    <xsl:otherwise>
      <xsl:variable name="fact">
        <xsl:call-template name="factorielle">
          <xsl:with-param name="n" select="$n - 1" />
        </xsl:call-template>
      </xsl:variable>
      <xsl:value-of select="$fact * $n" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Named templates play a role analogous to functions in traditional programming languages.

# Loops

```
<xsl:for-each select="child">
  <xsl:sort
    select="@age" order="ascending"
    data-type="number"/>

  <xsl:value-of select="name" />
  <xsl:text> is </xsl:text>
  <xsl:value-of select="@age" />
</xsl:for-each>
```

The use of `<xsl:for-each>` is more or less an alternative to the use of `<xsl:template>` / `<xsl:apply-templates>`.

`<xsl:sort>` may also be used as a direct child of an `<xsl:apply-templates>` element.

## Dynamic Elements

```
<xsl:element name="{concat('p',@age)}"
  namespace="http://ns">
  <xsl:attribute name="name">
    <xsl:value-of select="name" />
  </xsl:attribute>
</xsl:element>
```

```
<person age="12">
  <name>titi</name>
</person>
```

→

```
<p12
  name="titi"
  xmlns="http://ns" />
```

### Remark

The value of the `name` attribute is here an *attribute template*: this attribute normally requires a string, not an XPath expression, but XPath expressions between curly braces are evaluated. This is often used with literal result elements: `<toto titi="{ $var + 1 }"/>`. Literal curly braces must be doubled.

## Additional Functions

`document($s)` returns the **document node** of the document at the URL `$s`

`generate-id($s)` returns a **unique identifier string** for the first node of the nodeset `$s` in document order. Useful for testing the identity of two different nodes, or to generate HTML anchor names.

### Remark

*If `expr` is an XPath expression returning a nodeset (for instance, a variable reference, or a call to the `document()` function), `expr`, followed by any number of predicates, can be used as if it were a first step of a path expression.*

### Examples

```
document("toto.xml")/*, name($nodeset[@id=3])
```

## XSLT 1.0 Implementations

Large number of implementations.

**Browsers** All modern browsers (Internet Explorer, Firefox, Opera, Safari) include XSLT engines, used to process `xml-styleSheet` references. Also available via **JavaScript**, with various interfaces.

**libxslt** Free **C** library for XSLT transformations. Includes `xsltproc` command-line tool. **Perl** and **Python** wrappers exist.

**Sablotron** Free **C++** XSLT engine.

**Xalan-C++** Free **C++** XSLT engine.

**JAXP** **Java** API for Transformation. Common interface for various JAVA XSLT engines (SAXON, Xalan, Oracle...). Starting from JDK 1.4, a version of Xalan is bundled with Java.

**System.Xml** **.NET** XML and XSLT library.

**php-xslt** XSLT extension for **PHP**, based on Sablotron.

**4XSLT** Free XSLT **Python** library.

# References

- <http://www.w3.org/TR/xslt>
- *XML in a nutshell*, Eliotte Rusty Harold & W. Scott Means, O'Reilly
- *Comprendre XSLT*, Bernd Amman & Philippe Rigaux, O'Reilly

# Outline

- 1 Introduction
- 2 Preliminaries
- 3 XPath 1.0
- 4 XSLT 1.0
- 5 Beyond XSLT 1.0**
  - Limitations of XSLT 1.0
  - Extension Functions
  - XSLT 2.0

## Limitations of XSLT 1.0

- Impossible to **process a temporary tree** stored into a variable (with `<xsl:variable name="t"><toto a="3"/></xsl:variable>` ). Sometimes indispensable!
- **Manipulation of strings** is not very easy.
- **Manipulation of sequences** of nodes (for instance, for extracting all nodes with a distinct value), is awkward.
- Impossible to define in a portable way **new functions** to be used in XPath expressions. Using named templates for the same purpose is often verbose, since something equivalent to  $y = f(2)$  needs to be written as:

```
<xsl:variable name="y">
  <xsl:call-template name="f">
    <xsl:with-param name="x" select="2" />
  </xsl:call-template>
</xsl:variable>
```

## Extension Functions

XSLT allows for **extension functions**, defined in specific namespaces. These functions are typically written in a classical programming language, but the mechanism depends on the precise XSLT engine used. **Extension elements** also exist.

Once they are defined, such extension functions can be used in XSLT in the following way:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="http://exslt.org/math"
  version="1.0"
  extension-element-prefixes="math">

  ...

  <xsl:value-of select="math:cos($angle)" />
```

## EXSLT

EXSLT (<http://www.exslt.org/>) is a collection of extensions to XSLT which are portable across some XSLT implementations. See the website for the description of the extensions, and which XSLT engines support them (varies greatly). Includes:

`exsl:node-set` solves one of the main limitation of XSLT, by allowing to **process temporary trees** stored in a variable.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:exsl="http://exslt.org/common"
  version="1.0" extension-element-prefixes="exsl">

  ...

  <xsl:variable name="t"><toto a="3" /></xsl:variable>

  <xsl:value-of select="exsl:node-set($t)/*/a" />
```

`date` library for formatting **dates** and **times**

`math` library of **mathematical** (in particular, **trigonometric**)  
functions

`regexp` library for **regular expressions**

`strings` library for manipulating **strings**

...

Other extension functions outside EXSLT may be provided by each XSLT engine.

# XSLT 2.0

- W3C **Proposed Recommendation** 21 November 2006
- Like XQuery 1.0, uses **XPath 2.0**, a much more powerful language than XPath 1.0:
  - ▶ **Strong typing**, in relation with XML Schemas
  - ▶ **Regular expressions**
  - ▶ **Loop** and **conditional** expressions
  - ▶ Manipulation of **sequences** of nodes and values
  - ▶ ...
- New functionalities in XSLT 2.0:
  - ▶ Native processing of **temporary trees**
  - ▶ **Multiple** output documents
  - ▶ **Grouping** functionalities
  - ▶ **User-defined** functions
  - ▶ ...
- All in all, XSLT 2.0 stylesheets tend to be much more **concise** and **readable** than XSLT 1.0 stylesheets.

## Example XSLT 2.0 Stylesheet

Produces a list of each word appearing in a document, with their frequency.

(from *XSLT 2.0 Programmer's Reference*)

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <wordcount>
    <xsl:for-each-group group-by="." select=
      "for $w in tokenize(string(.), '\W+')
        return lower-case($w)">
      <word word="{current-grouping-key()}"
        frequency="{count(current-group())}"/>
    </xsl:for-each-group>
  </wordcount>
</xsl:template>

</xsl:stylesheet>
```

## XSLT 2.0 Implementations

Very few implementations as of January 2007.

**SAXON** **Java** and **.NET** implementation of XSLT 2.0 and XQuery 1.0. The full version is commercial (free evaluation version), but a GPL version is available without support of external XML Schemas.

**Oracle XML Developer's Kit** **Java** implementation of various XML technologies, including XSLT 2.0, XQuery 1.0, with full support of XML Schema. Less mature than SAXON.

# References

- <http://www.w3.org/TR/xpath20/>
- <http://www.w3.org/TR/xslt20/>
- *XPath 2.0 Programmer's Reference*, Michael Kay, Wrox
- *XSLT 2.0 Programmer's Reference*, Michael Kay, Wrox