# XSL 1.0 Quick Reference

## Intro

XSL includes XSLT and XPath.

## XSLT 1.0

### Object Types [§11.1, XPath §1]

| | |
|---|---|
| boolean | True or false |
| number | Floating-point number |
| string | UCS characters |
| node-set | Set of nodes selected by a path |
| Result tree fragment | XSLT only. Fragment of the result tree |

### Expression Context [§4, XPath §1]

Context node (a node)
Context position (a number)
Context size (a number)
Variable bindings in scope
Namespace declarations in scope
Function library

### XSLT Functions [§12, §15]

node-set **document**(object, node-set?)
node-set **key**(string, object)
string **format-number**(number, string, string?)
node-set **current**()
string **unparsed-entity-uri**(string)
string **generate-id**(node-set?)
object **system-property**(string)
boolean **element-available**(string)
boolean **function-available**(string)

### Processing

#### General

⇒ Select patterns return a node set (or value). Match patterns return a boolean.
⇒ Begins at the document root template.
⇒ Processing then proceeds from template rule to template rule ONLY VIA apply-templates.
⇒ A match pattern tests against a particular node. It sets that node, if matched, as the context node.
⇒ Relative Xpath expressions (those that don't begin with '/'), are always with respect to a context node.

#### Traversal Order

⇒ Traversal of the nodes via recursion proceeds as deep as possible along a branch to the leaf before moving back up ONLY ONE LEVEL to process the next child and thence as deep as possible along that branch.

Source:

```
<body>
  <p>
    <em>Emphatic
      <i>and italicised paragraph.</i>
    </em>
    <b>Some bold.</b>
  </p>
  <span>More writing</span>
</body>
```

Stylesheet:

```
<xsl:template match="*" xml:space="preserve">
  <tr class="element">
    <td><xsl:value-of select="name(.)" /></td>
  </tr>
  <xsl:apply-templates  select="*" />
</xsl:template>
```

Output

| |
|---|
| body |
| p |
| em |
| i |
| b |
| span |

Observe that after the leaf <i> it is <b> that is processed NOT .

### Template Rule Conflict Resolution

The more specific match patterns override (have a higher priority than) the more general ones.

| Examples | Priority |
|---|---|
| [Templates with lower import precedence] | It depends. |
| *<br>@*<br>node()<br>comments()<br>text()<br>processing-instruction() | -0.5 |
| user:*<br>@user:* | -0.25 |
| cool<br>user:cool<br>@cool<br>@user:cool | 0 |
| cool[@shirt] | 0.5 |
| <xsl:template priority="1" match="node()\|@*" | user specified. |

If templates have the same priority then the last in the stylesheet wins.

### Built-in Template Rules [§5.8]

Built in templates lie invisibly at the **top** of the document.

```
<xsl:template match="*|/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*|/" mode="m">
   <xsl:apply-templates mode="m"/>
</xsl:template>

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

<xsl:template
   match="processing-instruction()|comment()"/>
```

## XSLT Template

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

 <xsl:output omit-xml-declaration="yes" />

 <xsl:param name="nowLocalFriendly">Wed, 16 Jun 1999 11:02 (+1000 AUS Eastern Standard
Time)</xsl:param>

    <!-- Match: The document root -->
  <xsl:template match="/">
<!-- Keep the closing text tag indented flush for output formatting -->
    <xsl:text disable-output-escaping="yes">&lt;!DOCTYPE html PUBLIC "-//w3c//dtd xhtml
1.1//en" "http://www.w3.org/tr/xhtml11/dtd/xhtml11.dtd"&gt;
</xsl:text>
    <xsl:copy>
      <xsl:apply-templates select="*|comment()"/>
    </xsl:copy>
  </xsl:template>

  <!--  Identity transformation -->
  <xsl:template match="*">
    <xsl:element name="{name(.)}" >
      <xsl:for-each select="@*">
          <xsl:attribute name="{name(.)}" >
                <xsl:value-of select="."/>
          </xsl:attribute>
      </xsl:for-each>
      <xsl:apply-templates />
    </xsl:element>
  </xsl:template>

  <!--
      Match:   Elements with empty content, explicitly named, and regardles of their
namespace.
      Example Output:
                <br />
                <cool:myEmptyTag cool:feeling="nice" />
  -->
  <xsl:template match=
                    "*[local-name()='br']
                    | *[local-name()='meta']
                    | *[local-name()='link']
                    | *[local-name()='myEmptyTag']">
  <xsl:text disable-output-escaping="yes">&lt;</xsl:text>
  <xsl:value-of select="name(.)" />
  <xsl:for-each select="@*" >
    <xsl:text> </xsl:text>
    <xsl:value-of select="name(.)" />="<xsl:value-of select="."/>"</xsl:for-each>
  <xsl:text disable-output-escaping="yes"> /&gt;</xsl:text>
  </xsl:template>

  <xsl:template match="comment()">
   <xsl:copy /><xsl:text>&#10;</xsl:text>
  </xsl:template>
  <!-- ********** Website specific bellow here ******************* -->

  <!-- Make a match whether the element belongs to a namespace
       or belongs to no namespace -->
  <xsl:template match="*[local-name()='menu']" xml:space="preserve">
          <div class="menu" >
            <a href="index.htm" class="menuCommand">Home</a>
            <br />
            <a href="news.htm" class="menuCommand">News</a>
            <br />
          </div>
  </xsl:template>

  <xsl:template match="*[local-name()='websiteLastUpdated']" xml:space="preserve" >
              <div class="date" style="text-align: center;">
                Web Site Last Updated:  <xsl:value-of select="$nowLocalFriendly" />
                <br />
                <br />
              </div>
  </xsl:template>
</xsl:stylesheet>
```

## Stylesheet Element [§2.2]

<**xsl:stylesheet** version="1.0" id="id"
extension-element-prefixes="*tokens*"
exclude-result-prefixes="*tokens*"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> xsl:import*, top-level elements
</**xsl:stylesheet**>

**xsl:transform** is a synonym for xsl:stylesheet

## Combining Stylesheets [§2.6]

<**xsl:include href**="*uri-reference*"/>

<**xsl:import href**="*uri-reference*"/>
Any template rules **imported** into a style sheet have a lower import precedence than those that physically reside in that style sheet.

## Whitespace [§3.4]

<**xsl:strip-space elements**="*tokens*"/>

<**xsl:preserve-space elements**="*tokens*"/>

A single Line Feed in XML Makes for a new line:

```
<xsl:text>&#10;</xsl:text>
```

Also:

```
<xsl:template match="*[local-name()='menu']" xml:space="preserve">
...
</xsl:template>
```

```
Also:
<xsl:variable name="directoryLevel"
xml:space="no">
...
</xsl:variable>
```

## Defining Template Rules [§5.3]

<**xsl:template** match="*pattern*" name="*qname*"
priority="*number*" mode="*qname*">
xsl:param* followed by text, literal result elements
and/or XSL elements </**xsl:template**>

⇒ Select patterns return a node set (or value). Match patterns return a boolean.
⇒ Select patterns are with respect to the current context node of the match pattern.

## Applying Template Rules [§5.4]

<**xsl:apply-templates select**="*node-set-exp*"
**mode**="*qname*"/>
<**xsl:apply-templates select**="*node-set-exp*"
**mode**="*qname*">
(xsl:sort | xsl:with-param)* </**xsl:apply-templates**>

The following are equivalent.

```
<xsl:apply-templates  />
<xsl:apply-templates  select="node()"
/>
```

An apply-templates rule applies only to the immediate children of the context node not further descendants (at least not directly).

## Overriding Template Rules [§5.6]

<**xsl:apply-imports**/>

## Named Templates [§6]

<**xsl:call-template name**="*qname*"/>
<**xsl:call-template name**="*qname*">
xsl:with-param* </**xsl:call-template**>

## Namespace Alias [§7.1.1]

<**xsl:namespace-alias result-prefix**="*prefix*|*#default*"
**stylesheet-prefix**="*prefix*|*#default*"/>

## Creating Elements [§7.1.2]

<**xsl:element name**="{*qname*}"
namespace="{*uri-reference*}"
use-attribute-sets="*qnames*">...</**xsl:element**>

## Creating Attributes [§7.1.3]

<**xsl:attribute name**="{*qname*}"
namespace="{*uri-reference*}">...</**xsl:attribute**>

## Named Attribute Sets [§7.1.4]

<**xsl:attribute-set name**="*qname*"
use-attribute-sets="*qnames*">
xsl:attribute* </**xsl:attribute-set**>

## Creating Text [§7.2]

<**xsl:text** disable-output-escaping="yes|no">
#PCDATA </**xsl:text**>

## Processing Instructions [§7.3]

<**xsl:processing-instruction name**="{*ncname*}">
...</**xsl:processing-instruction**>

## Creating Comments [§7.4]

<**xsl:comment**>...</**xsl:comment**>

## Copying a Node[§7.5]

<**xsl:copy** use-attribute-sets="*qnames*">
...</**xsl:copy**>

⇒ `<xsl:copy />` copies the current node, including namespaces (if not defined at ancestor node), but not child nodes, nor child attributes.
⇒ Content inside `<xsl:copy></xsl:copy>` determines the children and attributes that are copied.

## Copy Tree Fragment and Values [§11.3]

<**xsl:copy-of select**="*expr*"/>

| Select Expression Returns | What is copied. |
|---|---|
| Result tree fragment | That result tree fragment |
| Node-set | Copy all nodes in the set in document order. For each node in the set copy node and all descendants (including attributes, namespaces, and element children). |
| Boolean, String, number | A string representation. |

## Generating Text [§7.6.1]

<**xsl:value-of select**="string-expr"
disable-output-escaping="yes|no"/>

## Attribute Value Templates [§7.6.2]

<*element attribute*="{expr}"/>

## Numbering [§7.7]

<**xsl:number** level="single|multiple|any"
count="*pattern*" from="*pattern*"
value="*number-expr*" format="{*string*}"
lang="{*nmtoken*}"
letter-value="{alphabetic|traditional}"
grouping-separator="{*char*}"
grouping-size="{*number*}" />

## Repetition [§8]

<**xsl:for-each select**="*node-set-expr*">
xsl:sort*, ...</**xsl:for-each**>

## Conditional Processing [§9]

<**xsl:if test**="*boolean-expr*">...</**xsl:if**>

<**xsl:choose**>
<**xsl:when test**="expr">...</**xsl:when**>+
<**xsl:otherwise**>...</**xsl:otherwise**>?
</**xsl:choose**>

## Sorting [§10]

<**xsl:sort** select="*string-expr*" lang="{*nmtoken*}"
data-type="{text|number|*qname-but-notncname*}"
order="{ascending|descending}"
case-order="{upper-first|lower-first}" />

## Variables and Parameters [§11]

<**xsl:variable name**="*qname*" select="*expr*"/>
<**xsl:variable name**="*qname*">...</**xsl:variable**>

<**xsl:param name**="*qname*" select="*expr*"/>
<**xsl:param name**="*qname*">...</**xsl:param**>

A variable in XSLT is a constant as in other languages. It has global or local scope.
An XSLT variable is global if it is declared as an immediate child element of the <xsl:stylesheet> element.

*Microsoft XML Core Services (MSXML) 4.0 > Using XSLT Variables and Parameters*

To use a param on an attribute with culry braces, {} in this XSLT specification (from W3C recommendation).

```
<xsl:param name="sort-order"
select="'descending'"/>
...
       <xsl:sort select="@*[name() =
$sort-column]" order="{$sort-order}"
```

Use a string value in the param (note extra apostrophies); use curly braces in your document. This is attribute value templates.

*Marrow 29 Aug 2002 > microsoft.public.xml > : Re: passing a parameter to <xsl:sort />*

To use a param on the select attribute, where attribute value templattes are not allowed.

```
<xsl:param name="sortField"
select="'title'" />
...
      <xsl:sort select="*[local-
name()=$sortField]" />
```

Use a string value in the param (note extra apostrophies); and a local-name() in the sort.

## Passing Parameters [§11.6]

<**xsl:with-param name**="expr" select="expr"/>
<**xsl:with-param name**="expr">...</**xsl:with-param**>

A parameter is a 'variable' in other languages. That is, it is only Xslt parameters that can change at runtime.

## Keys [§12.2]

<**xsl:key name**="*qname*" **match**="*pattern*"
**use**="expr"/>

## Number Formatting [§12.3]

<**xsl:decimal-format** name="qname"

```
    decimal-separator="char"
    grouping-separator="char" infinity="string"
    minus-sign="char" NaN="string"
    percent="char" per-mille="char"
    zero-digit="char" digit="char"
    pattern-separator="char"/>
```

### Messages [§13]

```
<xsl:message terminate="yes|no">
...</xsl:message>
```

### Fallback [§15]

```
<xsl:fallback>...</xsl:fallback>
```

### Output [§16]

```
<xsl:output
    method="xml|html|text|qname-but-not-ncname"
    version="nmtoken" encoding="string"
    omit-xml-declaration="yes|no"
    doctype-public="string" doctype-system="string"
    standalone="yes|no" indent="yes|no"
    cdata-section-elements="qnames"
    media-type="string"/>
```

# XPath 1.0

## Returns [XPath §1]

An XPath expression yields one of the following basic objects: node set; boolean; number; string

## Location Paths [XPath §2]

Optional '/', zero or more location steps, separated by '/'

A location path is absolute if it begins with a slash ("/") and relative (to the current context) otherwise.

## Location Steps [XPath §2.1]

axis::nodetest[predicate] <!-- zero or more predicates -->

## Axis Specifiers [XPath §2.2]

| | |
|---|---|
| ancestor:: | following-sibling:: |
| ancestor-or-self:: | namespace:: |
| attribute:: | parent:: |
| child:: | preceding:: |
| descendant:: | preceding-sibling:: |
| descendant-or-self:: | self:: |
| following:: | |

## Axis Principal node type [XPath §2.3]

⇒ Every axis has a **principal node type**. If an axis can contain elements, then the principal node type is element; otherwise, it is the type of the nodes that the axis can contain.
⇒ A node test * is true for any node of the principal node type.

| Axis | Meaning of * |
|---|---|
| Any axis other than an attribute:: axis or the namespace:: axis | element |
| attribute:: | attribute |
| namespace:: | namespace |

## Node Types [XPath §5]

| | |
|---|---|
| Root | Processing Instruction |
| Element | Comment |
| Attribute | Text |
| Namespace | |

## Main Node Tests [XPath §2.3]

| Test | Matches |
|---|---|
| / | Document root |
| node() | Element<br>Text<br>Comment<br>Processing-Instruction |
| @* | Attribute wildcard; selects all attributes regardless of name (or namespace) |
| name | |
| prefix:name | |
| * | Wildcard; selects principle node types regardless of node name. Usually selects any elements. |
| prefix:* | |
| text() | |
| comment() | |
| processing-instruction() | |
| processing-instruction(literal) | |

Shaded Node Tests together select the full contingent of node types (except namespaces).

```
*[local-name()='ElementName']
@*[local-name()='AttributeName']
```
Elements and attributes regardless of whether they have a namespace with: an explicit prefix; no prefix; or a default prefix .

```
*[namespace::*]
```
All elements regardless of namespace.
```
@*
```
All attributes regardless of namespace.

## Abbreviated Syntax for Location Paths

| | |
|---|---|
| (nothing) or ./ | child:: |
| @ | attribute:: |
| // | /descendant-or-self::node()/ |
| / | Document root |
| /*<br>(or * if Document root is context) | Root Element |
| . | Indicates the current context. |
| .. | The parent of the current context node. |
| * | Wildcard; selects principle node types regardless of node name. Usually selects any elements. |
| @* | Attribute wildcard; selects all attributes regardless of name. (or namespace) |
| : | Namespace separator; separates the namespace prefix from the element or attribute name. |
| ( ) | Groups operations to explicitly establish precedence. |
| [ ] | Applies a filter pattern. |
| [ ] | Subscript operator; used for indexing within a collection. Node sets start at 1. |

## Predicate [XPath §2.4]

[*expr*]

## Variable Reference [XPath §3.7]

$qname

## Literal Result Elements [§7.1.1]

Any element not in the xsl: namespace and not an extension element

## XPath Operators

Parentheses ( ) may be used for grouping.

## Node-sets [XPath §3.3]

| [expr] / //

## Booleans [XPath §3.4]

<=, <, >=, > =, != and or
(use < and > )

## Numbers [XPath §3.5]

-expr, *, div, mod +, -

## XPath Core Function Library

## Node Set Functions [XPath §4.1]

*number* **last**()
*number* **position**()
*number* **count**(*node-set*)
*node-set* **id**(*object*)
*string* **local-name**(*node-set*?)
*string* **namespace-uri**(*node-set*?)
*string* **name**(*node-set*?)

## String Functions [XPath §4.2]

*string* **string**(*object*?)
*string* **concat**(*string*, *string*, *string**)
*boolean* **starts-with**(*string*, *string*)
*boolean* **contains**(*string*, *string*)
*string* **substring-before**(*string*, *string*)
*string* **substring-after**(*string*, *string*)
*string* **substring**(*string*, *number*, *number*?)
*number* **string-length**(*string*?)
*string* **normalize-space**(*string*?)
*string* **translate**(*string*, *string*, *string*)

## Boolean Functions [XPath §4.3]

*boolean* **boolean**(*object*)
*boolean* **not**(*object*)
*boolean* **true**()
*boolean* **false**()
*boolean* **lang**(*string*)

## Number Functions [XPath §4.4]

*number* **number**(*object*?)
*number* **sum**(*node-set*)
*number* **floor**(*number*)
*number* **ceiling**(*number*)
*number* **round**(*number*)

## Examples

| Expression | Refers to |
|---|---|
| ./author | All <author> elements within the current context. Note that this is equivalent to the expression in the next row. |
| author | All <author> elements within the current context. |
| first.name | All <first.name> elements within the current context. |
| /bookstore | The document element (<bookstore>) of this document. |
| //author | All <author> elements in the document. |
| book[/bookstore/@specialty = @style] | All <book> elements whose style attribute value is equal to the specialty attribute value of the <bookstore> element at the root of the document. |
| author/first-name | All <first-name> elements that are children of an <author> element. |
| bookstore//title | All <title> elements one or more levels deep in the <bookstore> element (arbitrary descendants). Note that this is different from the expression in the next row. |
| bookstore/*/title | All <title> elements that are grandchildren of <bookstore> elements. |
| bookstore//book/excerpt//emph | All <emph> elements anywhere inside <excerpt> children of <book> elements, anywhere inside the <bookstore> element. |
| .//title | All <title> elements one or more levels deep in the current context. Note that this situation is essentially the only one in which the period notation is required. |
| author/* | All elements that are the children of <author> elements. |
| book/*/last-name | All <last-name> elements that are grandchildren of <book> elements. |
| */* | All grandchildren elements of the current context. |
| *[@specialty] | All elements with the specialty attribute. |
| @style | The style attribute of the current context. |
| price/@exchange | The exchange attribute on <price> elements within the current context. |
| price/@exchange/total | Returns an empty node set, because attributes do not contain element children. This expression is allowed by the XML Path Language (XPath) grammar, but is not strictly valid. |
| book[@style] | All <book> elements with style attributes, of the current context. |
| book/@style | The style attribute for all <book> elements of the current context. |
| @* | All attributes of the current element context. |
| ./first-name | All <first-name> elements in the current context node. Note that this is equivalent to the expression in the next row. |
| first-name | All <first-name> elements in the current context node. |
| author[1] | The first <author> element in the current context node. |
| author[first-name][3] | The third <author> element that has a <first-name> child. |
| my:book | The <book> element from the my namespace. |
| my:* | All elements from the my namespace. |
| @my:* | All attributes from the my namespace (this does not include unqualified attributes on elements from the my namespace). |
| *[.='gibbon'] \| marsupial[2] \| insect | Compound Location Step |
| book[position() <= 3] | The first three books (1, 2, 3). |
| book[last()] | The last <book> element of the current context node. |
| book/author[last()] | The last <author> child of each <book> element of the current context node. |
| (book/author)[last()] | The last `<author>` element from the entire set of `<author>` children of `<book>` elements of the current context node. |
| book[excerpt] | All `<book>` elements that contain at least one `<excerpt>` element child. |
| book[excerpt]/title | All `<title>` elements that are children of `<book>` elements that also contain at least one `<excerpt>` element child. |
| book[excerpt]/author[degree] | All `<author>` elements that contain at least one `<degree>` element child, and that are children of `<book>` elements that also contain at least one `<excerpt>` element. |
| book[author/degree] | All `<book>` elements that contain `<author>` children that in turn contain at least one `<degree>` child. |
| author[degree][award] | All `<author>` elements that contain at least one `<degree>` element child and at least one `<award> element child`. |
| author[degree and award] | All `<author>` elements that contain at least one `<degree>` element child and at least one `<award>` element child. |
| author[(degree or award) and title] | All `<author>` elements that contain at least one `<degree>` or `<award>` and at least one `<title>` as the children |
| author[degree and not(title)] | All `<author>` elements that contain at least one `<degree>` element child and that contain no `<title>` element children. |
| author[not(degree or award) and `title`] | All `<author>` elements that contain at least one `<title>` element child and contain neither `<degree>` nor `<award>` element children. |
| `author[last-name = "Bob"]` | All `<author>` elements that contain at least one `<last-name>` element child with the value `Bob`. |
| author[last-name[1] = "Bob"] | All `<author>` elements where the first `<last-name>` child element has the value `Bob`. Note that this is equivalent to the expression in the next row. |
| author[last-name [position()=1]= "Bob"] | All `<author>` elements where the first `<last-name>` child element has the value `Bob`. |
| degree[@from != "Harvard"] | All `<degree>` elements where the `from` attribute is not equal to `"Harvard"`. |
| author[. = "Matthew Bob"] | All `<author>` elements whose value is `Matthew Bob`. |
| author[last-name = "Bob" and ../price > 50] | All `<author>` elements that contain a `<last-name>` child element whose value is `Bob`, and a `<price>` sibling element whose value is greater than 50. |
| book[position() <= 3] | The first three books (1, 2, 3). |
| author[not(last-name = "Bob")] | All `<author>` elements that do no contain `<last-name>` child elements with the value `Bob`. |
| author[first-name = "Bob"] | All `<author>` elements that have at least one `<first-name>` child with the value `Bob`. |
| author[* = "Bob"] | all author elements containing any child element whose value is `Bob`. |
| author[last-name = "Bob" and first-name = "Joe"] | All `<author>` elements that has a |

| | |
|---|---|
| | <last-name> child element with the value Bob and a <first-name> child element with the value Joe. |
| price[@intl = "Canada"] | All <price> elements in the context node which have an intl attribute equal to "Canada". |
| degree[position() < 3] | The first two <degree> elements that are children of the context node. |
| p/text()[2] | The second text node in each <p> element in the context node. |
| ancestor::book[1] | The nearest <book> ancestor of the context node. |
| ancestor::book[author][1] | The nearest <book> ancestor of the context node and this <book> element has an <author> element as its child. |
| ancestor::author[parent::book][1] | The nearest <author> ancestor in the current context and this <author> element is a child of a <book> element. |
| (book/author)[last()] | The last <author> element from the entire set of <author> children of <book> elements of the current context node. |
| book[excerpt] | All <book> elements that contain at least one <excerpt> element child. |
| book[excerpt]/title | All <title> elements that are children of <book> elements that also contain at least one <excerpt> element child. |
| book[excerpt]/author[degree] | All <author> elements that contain at least one <degree> element child, and that are children of <book> elements that also contain at least one <excerpt> element. |
| book[author/degree] | All <book> elements that contain <author> children that in turn contain at least one <degree> child. |
| author[degree][award] | All <author> elements that contain at least one <degree> element child and at least one <award> element child. |
| author[degree and award] | All <author> elements that contain at least one <degree> element child and at least one <award> element child. |
| author[(degree or award) and title] | All <author> elements that contain at least one <degree> or <award> and at least one <title> as the children |
| author[degree and not(title)] | All <author> elements that contain at least one <degree> element child and that contain no <title> element children. |
| author[not(degree or award) and title] | All <author> elements that contain at least one <title> element child and contain neither <degree> nor <award> element children. |
| author[last-name = "Bob"] | All <author> elements that contain at least one <last-name> element child with the value Bob. |
| author[last-name[1] = "Bob"] | All <author> elements where the first <last-name> child element has the value Bob. Note that this is equivalent to the expression in the next row. |
| author[last-name [position()=1]= "Bob"] | All <author> elements where the first <last-name> child element has the value Bob. |
| degree[@from != "Harvard"] | All <degree> elements where the from attribute is not equal to "Harvard". |
| author[. = "Matthew Bob"] | All <author> elements whose value is Matthew Bob. |
| author[last-name = "Bob" and ../price > 50] | All <author> elements that contain a <last-name> child element whose value is Bob, and a <price> sibling element whose value is greater than 50. |
| book[position() <= 3] | The first three books (1, 2, 3). |
| author[not(last-name = "Bob")] | All <author> elements that do no contain <last-name> child elements with the value Bob. |
| author[first-name = "Bob"] | All <author> elements that have at least one <first-name> child with the value Bob. |
| author[* = "Bob"] | all author elements containing any child element whose value is Bob. |
| author[last-name = "Bob" and first-name = "Joe"] | All <author> elements that has a <last-name> child element with the value Bob and a <first-name> child element with the value Joe. |
| price[@intl = "Canada"] | All <price> elements in the context node which have an intl attribute equal to "Canada". |
| degree[position() < 3] | The first two <degree> elements that are children of the context node. |
| p/text()[2] | The second text node in each <p> element in the context node. |
| ancestor::book[1] | The nearest <book> ancestor of the context node. |
| ancestor::book[author][1] | The nearest <book> ancestor of the context node and this <book> element has an <author> element as its child. |
| ancestor::author[parent::book][1] | The nearest <author> ancestor in the current context and this <author> element is a child of a <book> element. |
| ancestor::book[author][1] | The nearest <book> ancestor of the context node and this <book> element has an <author> element as its child. |

# Key

## General

| | |
|---|---|
| **xsl:stylesheet** | Element |
| **version=** | Required attribute |
| version= | Optional attribute |
| {expr} | Attribute value template. Text between { and } is evaluated as an expression. Attribute value must evaluate to indicated attribute type. |
| … | Anything allowed in a template |
| \| | Separates alternative values |
| ? | Zero or one occurrences |
| * | Zero or more occurrences |
| + | One or more occurrences |
| #PCDATA | Character data |

## Attribute Value Types

| | |
|---|---|
| 1.0 | Literal value |
| *boolean-expr* | Expression returning boolean value |
| *char* | Single character |
| *expr* | Expression |
| *id* | XML name used as identifier |
| *ncname* | XML name not containing a colon (:) |
| *node-set-expr* | Expression returning a node set |
| *number-expr* | Expression returning a number |
| *pattern* | XSLT pattern |
| *prefix* | Namespace prefix |
| *qname* | Namespace-qualified XML name local part and optional prefix |
| *qname-but-notncname* | Namespace-qualified name comprising local part and prefix |
| *token* | Meaning varies with context. See Rec. |
| *uri-reference* | Reference to Universal Resource Identifier |

# Sources

⇒ XSL Transformations (XSLT)
Version 1.0 W3C Recommendation 16 November 1999
http://www.w3.org/TR/1999/REC-xslt-19991116

⇒ XML Path Language (XPath)
Version 1.0 W3C Recommendation 16 November 1999
http://www.w3.org/TR/1999/REC-xpath-19991116

⇒ Mulberry Technologies, Inc.
Original Quick Reference.

info@mulberrytech.com
http://www.mulberrytech.com

⇒ John Bentley formatted into Word and Made some updates to the Mulberry Quick Reference.

# Document Licence