XPath Reference

Composer: Edward Willemsen, 2007.

Contents

1. Introduction

An XML Path Language (Xpath) expression uses a path notation, like those used in URLs, for addressing parts of an XML document. The expression is evaluated to yield an object of the node-set, Boolean, number, or string type. For example, the expression book/author will return a node-set of the <author> elements contained in the <book> elements, if such elements are declared in the source XML document. In addition, an XPath expression can have predicates (filter expressions) or function calls. For example, the expression book[@type="Fiction"] refers to the <book> elements whose type attribute is set to "Fiction".

The following table summarizes some of the analogous features between URLs and XPath expressions.

URLs	XPath expressions
Hierarchy comprised of folders and files in a file system.	Hierarchy comprised of elements and other nodes in an XML document.
Files at each level have unique names. URLs always identify a single file.	Element names at each level might not be unique. XPath expressions identify a set of all the matching elements.
Evaluated relative to a particular folder, called the "current folder."	Evaluated relative to a particular node called the "context" for the expression.

As a XML syntax reminder:

<a href=”www.utools.nl”>uTools</a>

The <a></a> tags are elements and the ‘href’ is an attribute.

2. Context for XPath Expressions

The evaluation of an XPath expression depends on the context against which the expression operates. The context consists of the node against which the expression is evaluated and its associated environment, which includes the following:

- The position of the context node in the document order, relative to its siblings.
- The size of the context — that is, the number of siblings of the context node plus one.
- Variable bindings with which references to a variable can be resolved.
- A function library.
- The namespace declarations in scope for the expression.

To better appreciate the concept of context, consider a tree containing nodes. Asking for all nodes named X from the root of the tree returns one set of results, while asking for those nodes from a branch in the tree returns a different set of results. Thus, the result of an expression depends upon the context against which it executes.

XPath expressions can match specific patterns at one particular context, return the results, and perform additional operations relative to the context of the returned nodes. This gives XPath expressions extraordinary flexibility in searching throughout the document tree.

2.1. Basic XPath Expressions

The following are basic types of XPath expressions. Each type is described below.

- Current context
- Document root
- Root element
- Recursive descent
- Specific element

2.1.1. Examples

The following examples show some basic XPath expressions. More complex expressions are possible by combining these simple expressions together and by using the various XPath operators and special characters.

Current context

An expression prefixed with a period and forward slash (./) explicitly uses the current context as the context. For example, the following expression refers to all <author> elements within the current context:

```
./author
```

Note that this is equivalent to the following:

```
author
```

Document root

An expression prefixed with a forward slash (/) uses the root of the document tree as the context. For example, the following expression refers to the <bookstore> element at the root of this document:

```
/bookstore
```

Root element

An expression that uses a forward slash followed by an asterisk (/*) uses the root element as the context. For example, the following expression finds the root element of the document:

```
/*
```

Recursive descent

An expression that uses the double forward slash (//) indicates a search that can include zero or more levels of hierarchy. When this operator appears at the beginning of the pattern, the context is relative to the root of the document. For example, the following expression refers to all <author> elements anywhere within the current document:

```
//author
```

The .// prefix indicates that the context starts at the level in the hierarchy indicated by the current context.

Specific elements

An expression that starts with an element name refers to a query of the specific element, starting from the current context node. For example, the following expression refers to the <background.jpg> element within the <images> element in the current context node:

    images/background.jpg

The following expression refers to the collection of <book> elements within the <bookstore> elements in the current context node:

    bookstore/book

The following expression refers to all <first.name> elements in the current context node:

    first.name

> Note
> Element names can include the period character (.). These names can be used just like any other name.

Context in the DOM

When using XPath expressions with the Microsoft XML DOM, the context is the Node object whose selectNodes method or selectSingleNode method is called.

When using XPath directly from the DOM, you define the context from a particular node.

Context in XSLT

When using XPath directly from the XSLT, you define the context by the current node.

3. Operators and Special Characters

XPath expressions are constructed using the operators and special characters shown in the following table.

Operator	Comment
/	Child operator; selects immediate children of the left-side collection. When this path operator appears at the start of the pattern, it indicates that children should be selected from the root node.
//	Recursive descent; searches for the specified element at any depth. When this path operator appears at the start of the pattern, it indicates recursive descent from the root node.
.	Indicates the current context.
..	The parent of the current context node.
*****	Wildcard; selects all elements regardless of the element name.

@	Attribute; prefix for an attribute name.
@*	Attribute wildcard; selects all attributes regardless of name.
:	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.
+	Performs addition.
-	Performs subtraction.
div	Performs floating-point division according to IEEE 754.
*****	Performs multiplication.
mod	Returns the remainder from a truncating division.

This table does not include Boolean and set operators, which are listed in '*Boolean, Comparison, and Set Expressions*' or '*Set Operations*'.

Precedence order (from highest precedence to lowest) is defined as indicated in the following table.

Precedence	Character	Purpose
1	( )	Grouping
2	[ ]	Filters
3	/ //	Path operations

The group operator, (), is applicable only at the top-level path expression. For example, (//author/degree | //author/name) is a valid grouping operation, but //author/(degree | name) is not.

The filter pattern operators ([]) have a higher precedence than the path operators (/ and //). For example, the expression //comment()[3] selects all comments with an index equal to 3 relative to the comment's parent anywhere in the document. This differs from the expression (//comment())[3], which selects the third comment from the set of all comments relative to the parent. The first expression can return more than one comment, while the latter can return only one comment.

These operators and special characters are described in detail throughout this reference.

3.1. Path Operators

The collection of elements of a certain type can be determined using the path operators (/ and //). These operators take as their arguments a "left side" collection on which to perform the selection and a "right side" collection indicating which elements to select. The child operator (/) selects from immediate children of the left-side collection, while the descendant operator (//) selects from arbitrary descendants of the left-side collection. In effect, // can be considered a substitute for one or more levels of hierarchy.

Note that the path operators change the context as the query is performed. By stringing path operators together, users can traverse the document tree.

3.1.1. Examples

Expression	Refers to
author/first-name	All <first-name> elements within an <author> element of the current context node.
bookstore//title	All <title> elements one or more levels deep in the <bookstore> element (arbitrary descendants). Note that this is different from the following pattern, bookstore/*/title.
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.

3.2. Wildcard Character

An element can be referenced without using its name by substituting the wildcard (*) collection. The * collection refers to all elements that are children of the current context, regardless of the tag name.

3.2.1. Examples

Expression	Refers to
author/*	All element children of <author> elements.
book/*/last-name	All <last–name> elements that are grandchildren of <book> elements.
/	All grandchildren elements of the current context.
my:book	The <book> element from the *my* namespace.
my:*	All elements from the *my* namespace.

Note that the pattern *:book is not supported.

3.3. Attributes

XPath denotes attribute names with the @ symbol. Attributes and child elements are treated impartially, and capabilities are equivalent between the two types wherever possible.

Note
Attributes cannot contain child elements, so syntax errors occur when path operators are applied to attributes. In addition, you cannot apply an index to attributes because, by definition, no order is defined for attributes.

3.3.1.Examples

Expression	Refers to
@style	The style attribute of the current element context.
price/@exchange	The exchange attribute of <price> elements within the current context.
book/@style	The style attribute of all <book> elements.

Note that the following example is not valid, because an attribute cannot have any children.

```
price/@exchange/total
```

3.4. Finding Multiple Attributes

All attributes of an element can be returned using @*. This is potentially useful for applications that treat attributes as fields in a record.

3.4.1. Examples

Expression	Refers to
@*	All attributes of the current context node.
@my:*	All attributes from the *my* namespace. This does not include unqualified attributes on elements from the *my* namespace.

Note that the pattern @*:title is not supported.

4. XPath Collections

Collections returned by XPath queries preserve document order, hierarchy, and identity, to the extent that these are defined. That is, a collection of elements is returned in document order without repeated elements. Because by definition attributes are unordered, there is no implicit order to attributes returned for a specific element.

The collection of all elements with a certain tag name is expressed using the tag name itself. This can be qualified by showing that the elements are selected from the current context by using a period and forward slash (./), but the current context is used by default and does not have to be noted explicitly.

4.1. Examples

Expression	Refers to
./first-name	All <first-name> elements. Note that this expression is equivalent to the expression that follows.
first-name	All <first-name> elements.

4.2. Indexing into a Collection

XPath expressions make it easy to query a specific node within a set of nodes. Simply enclose the index ordinal within square brackets. The ordinal is 1-based (the first element is number 1).

The square bracket characters ([]) have higher precedence than the slash characters (/ and //). (For more information see Operators and Special Characters).

4.2.1. Examples

Expression	Refers to
author[1]	The first <author> element.
author[first-name][3]	The third <author> element that has a <first-name> child element.

Note that indexes are relative to the set being filtered. Consider, for example, the following data.

```
<x>
  <y/>
  <y/>
</x>
<x>
  <y/>
  <y/>
</x>
```

The following table shows how to select specific <x> and <y> elements.

Expression	Refers to
x/y[1]	The first <y> inside each <x>.
(x/y)[1]	The first <y> from the entire set of <y> elements within <x> elements.
x[1]/y[1]	The first <y> inside the first <x>.

The examples above are simple references to XPath collections that use implied defaults, such as the child:: axis. For this axis, the collection of child nodes is indexed in forward document order.

For other axes, such as ancestor::, use the axis name explicitly in your XPath expression. For this axis, the collection of ancestors is indexed in reverse document order. Consider this example from the previous table:

```
x/y[1]
```

This expression is equivalent to this one:

```
x/child::y[1]
```

Both expressions mean "for each <x> element, select the first child element named <y>."

The following example uses the same syntax.

```
x/ancestor::y[1]
```

This example translates to "for each <x> element, select the first ancestor element (in reverse-document order) named <y>". The syntax is the same, but the order is reversed.

4.3. Finding the Last Element in a Collection

The last() function returns True for the last element in a collection. Note that last is relative to the parent node.

4.3.1. Examples

Expression	Refers to
book[last()]	The last <book> element.
book/author[last()]	The last <author> element inside each <book> element.
(book/author)[last()]	The last <author> element from the entire set of <author> elements inside <book> elements.

4.4. Grouping

Parentheses can be used to group collection operators for clarity or where the normal precedence is inadequate to express an operation. Grouping operators can be used in any filter expressions (predicates), such as author[(degree or award)and publication]. They can also be used in the top-level step expression, such as (book|magazine) or (author/degree | book/award). They cannot be applied to lower-level step expressions. For example, author/(degree | award)is not valid.

4.4.1. Examples

Expression	Refers to
(book/author)	All <author> elements that are child elements of any <book> element from the current context node.
author[(degree or award) and publication]	All <author> elements that contain at least one <degree> or <award> element and at least one <publication> element.

5. Filters and Filter Patterns

Constraints and branching can be applied to any collection by adding a filter clause, [pattern], to the collection. The filter is analogous to the SQL WHERE clause. The filter contains a pattern within it called the filter pattern. The filter pattern evaluates to a Boolean value and is tested for each element in the collection. Any elements in the collection failing the filter pattern test are omitted from the result collection.

For convenience, if a collection is placed within the filter, a Boolean TRUE is generated if the collection contains any members and a FALSE is generated if the collection is empty. An expression such as author/degree implies a collection-to-Boolean conversion function that evaluates to TRUE if there exists an <author> element with a child element named <degree>.

Note that any number of filters can appear at a given level of an expression. Empty filters are not allowed.

Filters are always evaluated with respect to a context. In other words, the expression book[author] means that for every <book> element that is found, test whether it has an <author> child element. Likewise, book[author = 'Bob'] means that for every <book> element that is found, test whether it has an <author> child element with the value Bob. One can examine the value of the context as well by using the period (.) character. For example, book[. = 'Trenton'] means that for every book that is found in the current context, test whether its value is Trenton.

5.1. Examples

Expression	Refers to
book[excerpt]	All <book> elements that contain at least one <excerpt> element.
book[excerpt]/title	All <title> elements inside <book> elements that contain at least one <excerpt> element.
book[excerpt]/author[degree]	All <author> elements that contain at least one <degree> element, and are inside of <book> elements that contain at least one <excerpt> element.
book[author/degree]	All <book> elements that contain at least one <author> element with at least one <degree> child element.
book[excerpt][title]	All <book> elements that contain at least one <excerpt> element and at least one <title> element.

6. Boolean, Comparison, and Set Expressions

Filter patterns can contain Boolean expressions, comparison expressions, and set expressions. Shortcuts listed in the following table represent alternative symbols that are provided in this XSL Transformations (XSLT) implementation. This documentation discusses these expression operators.

Operator	Description
and	Logical-and
or	Logical-or
not()	Negation
=	Equality
!=	Not equal
&lt; *	Less than
&lt;= *	Less than or equal
&gt; *	Greater than
&lt;= *	Greater than or equal
|	Set operation; returns the union of two sets of nodes

* Extended XPath method

The World Wide Web Consortium (W3C) syntax for operator keywords uses white space and other separators rather than the dollar sign character ($) used in version 2.5. In the W3C syntax, a binary keyword of the form $xxx$ can be expressed as wsxxxws, where ws refers to a token terminator that can be white space, single quote characters ('), or double quote characters ("). Unary operators such as not() use functional notation. Although the Microsoft implementation supports both syntaxes, it is recommended that the W3C syntax be used for future compatibility.

Precedence order (from highest to lowest) for comparison operators and Boolean operators is shown in the following table.

Order	Operator	Type
1	( )	Grouping
2	[ ]	Filters
3	/	Path operations

	//	

4	< or &lt;
	<= or &lt;=
	> or &gt;
	>= or &gt;=
		Comparisons

5	= !=	Comparisons
6	|	Union
7	not()	Boolean not
8	And	Boolean and
9	Or	Boolean or

When the operators are used in an XML document, such as an XSLT style sheet, the < and > tokens must be escaped as &lt; and &gt;, respectively. For example, the following XSLT instruction invokes an XSLT template rule on all <book> elements whose <price> element has a numeric value less than or equal to 10.

```
<xsl:apply-templates select="book[price &lt;= 10]"/>
```

When an XPath expression is used with DOM, the < and > operators need not to be escaped. For example, the following JScript statement selects all <book> elements whose <price> element has a numeric value less than or equal to 10.

```
var cheap_books = dom.selectNodes("book[price <= 10]");
```

Boolean expressions can match all nodes of a particular value or all nodes with nodes in particular ranges. The following is an example of a Boolean expression that returns false.

```
1 &gt;= 2
```

Operators are case-sensitive.

6.1. Logical-and and Logical-or

The Boolean operators and and or perform logical-and and logical-or operations, respectively. These operators, in conjunction with grouping parentheses, can be used to build sophisticated logical expressions.

6.1.1. Examples

Expression	Refers to
author[degree and award]	All <author> elements that contain at least one <degree> element and at least one <award> element.
author[(degree or award) and publication]	All <author> elements that contain at least one <degree> or <award> element, and at least one <publication> element.

6.2. Boolean not

The Boolean not operator negates the value of an expression within a filter pattern.

6.2.1. Examples

Expression	Refers to
author[degree and not(publication)]	All <author> elements that contain at least one <degree> element, but contain no <publication> elements
author[not(degree or award) and publication]	All <author> elements that contain at least one <publication> element, but do not contain any <degree> elements or <award> elements.

XML File (test.xml)

```
<?xml version="1.0"?>
<test>
  <x a="1">
    <x a="2" b="B">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
  <x a="2">
    <y>y2</y>
  </x>
  <x a="3">
    <y>y3</y>
  </x>
</test>
```

XSLT File (test.xsl)

The following XSLT stylesheet selects all the <x> elements without any attributes.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>

<!-- suppress text nodes not covered in subsequent template rule -->
<xsl:template match="text()"/>
```

```
<xsl:template match="*">
  <xsl:element name="{name()}">
    <xsl:apply-templates select="*|@*"/>
    <xsl:if test="text()">
      <xsl:value-of select="."/>
    </xsl:if>
  </xsl:element>
</xsl:template>

<xsl:template match="@*">
  <xsl:attribute name="{name()}">
    <xsl:value-of select="."/>
  </xsl:attribute>
</xsl:template>

<xsl:template match="/test">
 <xsl:apply-templates select="//x[not(@*)] "/>
</xsl:template>
</xsl:stylesheet>
```

Output

The transformation, when applied to the XML file given above yields the following result:

```
<x>
  <y>y31</y>
  <y>y32</y>
</x>
```

7. Comparisons

To compare two objects in XPath, use the = sign to test for equality, or use != to test for inequality.

For a comparison operation, exactly two operands must be supplied. Comparisons are then made by evaluating each operand, and converting them as needed, so they are of the same type. This is done according to the process described below, in "Order of Precedence For Comparisons".

All elements and attributes are strings, but are automatically cast as integer values for numeric comparisons. Literal numeric values are cast to long or double types during comparison operations, as shown in the following table.

For information about &lt; and other binary comparison operators, see "Binary Comparison Operators", shown further in this document.

Literal type	Comparison	Example
String	text(lvalue) op text(rvalue)	a &lt; GGG
Integer	(long) lvalue op (long) rvalue	a &lt; 3
Real	(double) lvalue op (double) rvalue	a &lt; 3.1

Single or double quotation marks can be used for string delimiters in expressions. This makes it easier to construct and pass patterns from within scripting languages.

For more information about how comparisons are performed using XPath, see section 3.4 ("Booleans") of the XML Path Language (XPath) Version 1.0 (W3C Recommendation 16 November 1999) at www.w3.org/TR/xpath.

7.1. Examples

Expression	Refers to
author[last-name = "Bob"]	All <author> elements that contain at least one <last-name> element with the value Bob.
author[last-name[1] = "Bob"]	All <author> elements whose first <last-name> child element has the value Bob.
author/degree[@from != "Harvard"]	All <author> elements that contain <degree> elements with a from attribute that is not equal to "Harvard".
author[last-name = /editor/last-name]	All <author> elements that contain a <last-name> element that is the same as the <last-name> element inside the <editor> element under the root element.
author[. = "Matthew Bob"]	All <author> elements whose string value is Matthew Bob.

7.2. Order of Precedence for Comparisons

Comparisons with regard to data types obey the order of precedence.

If at least one operand is a Boolean, each operand is first converted to a Boolean.

Otherwise, if at least one operand is a number, each operand is first converted to a number.

Otherwise, if at least one operand is a date, each operand is first converted to a date.

Otherwise, both operands are first converted to strings.

7.3.Binary Comparison Operators

A set of binary comparison operators compares numbers and returns Boolean results. The &lt;, &lt;=, &gt;, and &gt;= operators are used for less than, less than or equal, greater than, and greater than or equal, respectively. Single or double quotation marks can be used for string delimiters in expressions. This makes it easier to construct and pass patterns within scripting languages.

Note that these comparison operators work only with numbers. You can compare strings for equality, but if you want to compare strings to determine which comes first in sort order, you need to use the Microsoft XPath Extension Functions (http://msdn2.microsoft.com/en-us/library/ms256453.aspx).

7.3.1. Examples

Expression	Refers to
author[last-name = "Bob" and price &gt; 50]	All <author> elements that contain a <last-name> element with the value Bob, and a <price> element with a value greater than 50.
degree[@from != "Harvard"]	All <degree> elements with a from attribute that is not equal to "Harvard".
book[position() &lt;= 3]	The first three <book> elements (1, 2, 3) in the XML file.

XML File (test.xml)

```
<?xml version="1.0"?>
<test>
  <x a="1">
    <x a="2" b="B">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
  <x a="2">
    <y>y2</y>
  </x>
  <x a="3">
    <y>y3</y>
  </x>
</test>
```

XSLT File (test.xsl)

The following XSLT style sheet selects all the <x> elements that are the first of their siblings in the document order.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>
```

```
<!-- Suppress text nodes not covered in subsequent template rule. -->
<xsl:template match="text()"/>
<xsl:template match="*">
  <xsl:element name="{name()}">
    <xsl:apply-templates select="*|@*"/>
    <xsl:if test="text()">
      <xsl:value-of select="."/>
    </xsl:if>
  </xsl:element>
</xsl:template>

<xsl:template match="@*">
  <xsl:attribute name="{name()}">
    <xsl:value-of select="."/>
  </xsl:attribute>
</xsl:template>

<xsl:template match="/test">
 <xsl:apply-templates select="//x[position() = 1 ] "/>
</xsl:template>
</xsl:stylesheet>
```

Formatted Output

The transformation applied to the XML file above yields the following result:

```
<x a="1">
 <x a="2" b="B">
   <x>
     <y>y31</y>
     <y>y32</y>
   </x>
  </x>
</x>
<x a="2" b="B">
  <x>
    <y>y31</y>
    <y>y32</y>
  </x>
</x>
<x>
  <y>y31</y>
  <y>y32</y>
</x>
```

8. Set Operations

XML Path Language (XPath) supports the set operation |.

8.1. Union (|) Operator

The |, or union, operator returns the union of its two operands, which must be node-sets. For example, //author | //publisher returns a node-set that combines all the //author nodes and all the //publisher nodes. Multiple union operators can be chained together to combine multiple node-sets. For example, //author | //publisher | //editor | //book-seller returns a node-set containing all //author, //publisher, //editor, and //book-seller elements. The union operator preserves document order and does not return duplicates.

Expression	Refers to
first-name | last-name	A node set containing <first-name> and <last-name> elements in the current context.
(bookstore/book | bookstore/magazine)	A node set containing <book> or <magazine> elements inside a <bookstore> element.
book | book/author	A node set containing all <book> elements and all <author> elements within <book> elements.
(book | magazine)/price	The node set containing all <price> elements of either <book> or <magazine> elements.

The following example illustrates the effect of the union operator.

XML File (test.xml)

```
<?xml version="1.0"?>
<test>
  <x a="1">
    <x a="2" b="B">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
</test>
```

XSLT File (test.xsl)

The following XSLT style sheet selects all the <x> elements whose a attribute is equal to 2, plus those <x> elements that have no attributes.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>
  <!-- Suppress text nodes not covered in subsequent template rule. -->
  <xsl:template match="text()"/>

  <!-- Handles a generic element node. -->

  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:apply-templates select="*|@*" />
      <xsl:if test="text()">
        <xsl:value-of select="."/>
      </xsl:if>
    </xsl:element>
  </xsl:template>

  <!-- Handles a generic attribute node. -->
  <xsl:template match="@*">
    <xsl:attribute name="{name()}">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>

  <xsl:template match="/test">
    <xsl:apply-templates select="//x[@a=2] | //x[not(@*)]"/>
  </xsl:template>
</xsl:stylesheet>
```

The transformation yields the following result:

```
<x a="2" b="B">
  <x>
    <y>31</y>
    <y>y32</y>
  </x>
</x>
<x>
  <y>y31</y>
  <y>y32</y>
</x>
```

8.2. Precedence

Precedence order (from highest precedence to lowest) between Boolean and comparison operators is shown in the following table.

Order	Operator	Type
1	( )	Grouping
2	[ ]	Filters

3	/ //	Path operations

4	&lt;
	&lt;=
	&gt;
	&gt;=
		Comparisons

5	= !=	Comparisons
6	|	Union
7	not()	Boolean not
8	and	Boolean and
9	or	Boolean or

The following example illustrates the effect of the operator precedence listed above.

XML File (test.xml)

```
<?xml version="1.0"?>
<test>
  <x a="1">
    <x a="2" b="B">
      <x>
        <y>y31</y>
        <y>y32</y>
      </x>
    </x>
  </x>
  <x a="1">
    <x a="2">
      <y>y21</y>
      <y>y22</y>
    </x>
  </x>

  <x a="1">
    <y>y11</y>
    <y>y12</y>
  </x>

  <x>
    <y>y03</y>
    <y>y04</y>
  </x>
</test>
```

Basic XSLT File (test.xsl)

We will use this basic XSLT file as a starting point for the series of illustrations that follow.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>

  <!-- Suppress text nodes not covered in subsequent template rule. -->
  <xsl:template match="text()"/>
 <!-- Handles a generic element node. -->
  <xsl:template match="*">
    <xsl:element name="{name()}">
      <xsl:apply-templates select="*|@*" />
      <xsl:if test="text()">
        <xsl:value-of select="."/>
      </xsl:if>
    </xsl:element>
  </xsl:template>

  <!-- Handles a generic attribute node. -->
  <xsl:template match="@*">
    <xsl:attribute name="{name()}">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>
</xsl:stylesheet>
```

8.3. Examples

Case 0. Test run

You can add the following template-rule to the XSLT style sheet.

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

This will produce an XML document identical to the original one, without the <?xml version="1.0"?> processing instruction.

The following cases show different ways of writing this template rule. The point is to show the order in which the XPath operators bind to an element.

Case 1: () binds tighter than []

The following template rule selects the first <y> element in the document order, from all the <y> elements in the source document.

```
<xsl:template match="/test">
    <xsl:apply-templates select="(//y)[1]"/>
 </xsl:template>
```

The result is as follows:

```
<y>y31</y>
```

Case 2: [] binds tighter than / or //

The following template rule selects all the <y> elements that are the first among their siblings.

```
<xsl:template match="/test">
  <xsl:apply-templates select="//y[1]"/>
</xsl:template>
```

The result is as follows:

```
<y>y31</y>
<y>y21</y>
<y>y11</y>
<y>y03</y>
```

Case 3: and, not

The following template rule selects all the <x> elements that have no <x> child elements, that have an <x> parent element, and that do not have any attributes.

```
<xsl:template match="/test">
  <xsl:apply-templates select=
     "//x[./ancestor::*[name()='x'] and *[name()!='x'] and not(@*)]"/>
</xsl:template>
```

The result is a single <x> element, listed below with its children:

```
<x>
  <y>y31</y>
  <y>y32</y>
</x>
```

Case 4: or, and, not

The following template rule selects each <x> elements that is a child of an <x> element; or, that is not a parent of an <x> element and has no attributes.

```
<xsl:template match="/test">
  <xsl:apply-templates select=
   "//x[./ancestor::*[name()='x'] or *[name()!='x'] and not(@*)]"/>
</xsl:template>
```

The result is a node set containing the following <x> elements, listed below with its children:

```
<x a="2" b="B">
  <x>
    <y>y31</y>
    <y>y32</y>
  </x>
</x>
<x>
  <y>y31</y>
  <y>y32</y>
</x>
<x a="2">
  <y>y21</y>
  <y>y22</y>
</x>
<x>
  <y>y03</y>
  <y>y04</y>
</x>
```

Case 5: and, or, not

The following template rule selects each <x> element that is a child of an <x> element but not a parent of an <x> element; or, that has no attributes.

```
<xsl:template match="/test">
  <xsl:apply-templates select=
   "//x[./ancestor::*[name()='x'] and *[name()!='x'] or not(@*)]"/>
</xsl:template>
```

The result is a node set containing the following <x> elements, listed below with its children:

```
<x>
  <y>y31</y>
  <y>y32</y>
</x>
<x a="2">
  <y>y21</y>
  <y>y22</y>
</x>
<x>
  <y>y03</y>
  <y>y04</y>
</x>
```

9. Location Paths

A location path is an XPath expression used for selecting a set of nodes relative to the context node. The evaluation of a location path expression results in a node-set containing the nodes specified by the location path. A location path can recursively contain expressions used to filter sets of nodes.

Syntactically, a location path consists of one or more location steps, each separated by a forward slash (/):

```
locationstep/locationstep/locationstep
```

Each location step in turn selects a set of nodes relative to the context node — that is, to the node selected by the preceding location step. A location path expressed this way is a relative location path. An absolute location path starts from the root element:

```
/locationstep/locationstep/locationstep
```

In a location path, location steps are evaluated from left to right. The leftmost location step selects a set of nodes relative to the context node. These nodes then become the context used to process the next location step. This processing of steps and updating of the context node repeats itself until all the location steps have been processed.

A location path can be unabbreviated or abbreviated.

In an unabbreviated location path, a location step has the following syntax:

```
axis::node-test[predicate]
```

In this syntax, axis specifies how the nodes selected by the location step are related to the context node; node-test specifies the node type and expanded name of the nodes selected by the location step; and predicate is a filter expression to further refine the selection of nodes in the location step. Predicates are optional. In this case, a location step consists of axis:: and node-test only. The following table provides some examples.

Unabbreviated Location Path	Description
child::para[last()]	Selects the last <para> element of the context node.
parent::para	Selects the <para> element that is the parent of the context node.
child::text()	Selects all text node children of the context node.
child::div/child::para	Selects the <para> child elements of the <div> element that is a child of the context node.

In an abbreviated location path, the axis specifier, axis::, is not expressed explicitly in a location step, but implied by a set of shortcuts instead. The following table provides some examples.

Abbreviated Location Path	Description
para	Selects the <para> elements of the context node.
../para	Selects the <para> element that is the parent of the context node.
text()	Selects all text node children of the context node.
./div/para	Selects the <para> element children of the <div> element children of the context node.

The following is a summary of some of the abbreviations:

Unabbreviated	Abbreviated
child::*	*
attribute::*	@*
/descendant-or-self::node()	//
self::node()	.
parent::node()	..

10. XPath Examples

This topic reviews the syntax examples that appear throughout the XPath Reference. All are based on the Sample XML File for XPath Syntax (inventory.xml). For an example of using an XPath expression in a test file, see "Example of Unions (|)", at the bottom of this topic.

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).

Note that indexes are relative to the parent. Consider the following data:

```
<x>
  <y/>
  <y/>
</x>
<x>
  <y/>
  <y/>
</x>
```

Expression	Refers to
x/y[1]	The first <y> child of each <x>. This is equivalent to the expression in the next row.
x/y[position() = 1]	The first <y> child of each <x>.
(x/y)[1]	The first <y> from the entire set of <y> children of <x> elements.
x[1]/y[2]	The second <y> child of the first <x>.

The remaining examples refer to the Sample XML file for XPath.

Expression	Refers to
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 publication]	All <author> elements that contain at least one <degree> or <award> and at least one <publication> as the children.
author[degree and not(publication)]	All <author> elements that contain at least one <degree> element child and that contain no <publication> element children.
author[not(degree or award) and publication]	All <author> elements that contain at least one <publication> 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 &gt; 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() &lt;= 3]	The first three books (1, 2, 3).
author[not(last-name = "Bob")]	All <author> elements that do not 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 have 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() &lt; 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.

10.1. Example of Unions (|)

To demonstrate the union operation, we use the following XPath expression:

```
x | y/x
```

selects all the <x> elements whose values are green or blue in the following XML file:

XML File (data1.xml)

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="union.xsl"?>
<root>
  <x>green</x>
  <y>
    <x>blue</x>
    <x>blue</x>
  </y>
  <z>
    <x>red</x>
    <x>red</x>
  </z>
  <x>green</x>
</root>
```

XSLT File (union.xsl)

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

<xsl:template match="root">
  <xsl:for-each select="x | y/x">
    <xsl:value-of select="."/>,
    <xsl:if test="not(position()=last())">,</xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Formatted Output

```
green,blue,blue,green
```

Processor Output

```
<?xml version="1.0" encoding="UTF-16"?>green,blue,blue,green
```

11. XPath Functions

You can use XML Path Language (Xpath) functions to refine XPath queries and enhance the programming power and flexibility of XPath.

The functions are divided into the following groups.

Function	Description
Node-Set	Takes a node-set argument, returns a node-set, or returns/provides information about a particular node within a node-set.
String	Performs evaluations, formatting, and manipulation on string arguments.
Boolean	Evaluates the argument expressions to obtain a Boolean result.
Number	Evaluates the argument expressions to obtain a numeric result.
Microsoft XPath Extension Functions	Microsoft extension functions to XPath that provide the ability to select nodes by XSD type. Also includes string comparison, number comparison, and date/time conversion functions.

Each function in the function library is specified using a function prototype that provides the return type, function name, and argument type. If an argument type is followed by a question mark, the argument is optional; otherwise, the argument is required. Function names are case-sensitive.

11.1. Node-Set Functions

Node-set functions take a node-set argument. They return a node-set, or information about a particular node within a node-set.

Node-Set Function	Description
count	Returns the number of nodes in the node-set argument.
id	Selects elements by their unique ID.
last	Returns a number equal to context size of the expression evaluation context.
local-name	Returns the local part of the expanded name of the node in the node-set argument that is first in document order.
name	Returns a string containing a QName representing the expanded name of the node in the node-set argument that is first in document order.
namespace-uri	Returns the namespace Uniform Resource Identifier (URI) of the expanded name of the node in the node-set argument that is first in document order.
position	Returns the index number of the node within the parent.

11.2. String Functions [XPath]

String functions are used to evaluate, format, and manipulate string arguments, or to convert an object to a string.

String Functions	Description
concat	Returns the concatenation of the arguments.
contains	Returns true if the first argument string contains the second argument string; otherwise returns false.
normalize-space	Returns the argument string with the white space stripped.
starts-with	Returns true if the first argument string starts with the second argument string; otherwise returns false.
string	Converts an object to a string.
string-length	Returns the number of characters in the string.
substring	Returns the substring of the first argument starting at the position specified in the second argument and the length specified in the third argument.
substring-after	Returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string.
substring-before	Returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string.
translate	Returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string.

11.3. Boolean Functions

The XML Path Language (XPath) syntax supports Boolean functions that return strings or numbers, and can be used with comparison operators in filter patterns.

Boolean Functions	Description
boolean	Converts the argument to a Boolean.
false	Returns false.
lang	Returns true if the xml:lang attribute of the context node is the same as the argument string.
not	Returns true if the argument is false, otherwise, false.
true	Returns true.

11.4. Number Functions

The XML Path Language (XPath) syntax supports Number functions that return strings or numbers and can be used with comparison operators in filter patterns.

Number Functions	Description
ceiling	Returns the smallest integer that is not less than the argument.
floor	Returns the largest integer that is not greater than the argument.
number	Converts the argument to a number.
round	Returns an integer closest in value to the argument.
sum	Returns the sum of all nodes in the node-set. Each node is first converted to a number value before summing.

11.5. Microsoft XPath Extension Functions

MSXML provides a number of extension functions to offer additional features beyond those specified in the XPath Version 1.0 specification. Some of these extension functions enable manipulations of nodes based on their XSD data types. Others provide some popular utilities, such as lexicographical comparison of strings, formatting times and dates, converting date/time to Coordinated Universal Time units, etc.

The names of extended functions must be of qualified name consisting of a namespace URI (or its proxy), a colon, and a local part. Microsoft XPath extension functions typically sport a ms prefix that has been associated with the namespace URI ("urn:schemas-microsoft-com:xslt") for the Microsoft extension functions.

11.5.1. XPath Extension Functions for XSD Support

Function	Description
ms:type-is	Compares the current node's data type against the specified node type.
ms:type-local-name ([node-set])	Returns the nonqualified name of the XSD type of the current node or the first node (in document order) in the provided node-set.
ms:type-namespace-uri ([node-set])	Returns the namespace URI associated with the XSD type of a current node or the first node (in document order) in the provided node-set.
ms:schema-info-available	Returns true if XSD information is available for a current node.

11.5.2. XPath Extension Functions of Miscellaneous Utilities

Function	Description
ms:string-compare	Performs lexicographical string comparison.
ms:utc	Converts the prefixed date/time related values into Coordinated Universal Time and into a fixed (normalized) representation that can be sorted and compared lexicographically.
ms:namespace-uri	Resolves the prefix part of a qualified name into a namespace URI.
ms:local-name	Returns the local name part of a qualified name by stripping out the namespace prefix.
ms:number	Takes a string argument in XSD format and converts it into an XPath number.
ms:format-date	Converts standard XSD date formats to characters suitable for output.
ms:format-time	Converts standard XSD time formats to characters suitable for output.

12. Source

http://msdn2.microsoft.com/en-us/library/ms256471.aspx

http://msdn2.microsoft.com/en-us/library/ms256199.aspx

http://msdn2.microsoft.com/en-us/library/ms256122.aspx

http://msdn2.microsoft.com/en-us/library/ms256090.aspx

http://msdn2.microsoft.com/en-us/library/ms256060.aspx

http://msdn2.microsoft.com/en-us/library/ms256081.aspx

http://msdn2.microsoft.com/en-us/library/ms256135.aspx

http://msdn2.microsoft.com/en-us/library/ms256074.aspx

http://msdn2.microsoft.com/en-us/library/ms256039.aspx

http://msdn2.microsoft.com/en-us/library/ms256086.aspx

http://msdn2.microsoft.com/en-us/library/ms256138.aspx

Other Resources

For more information, see the XML Path Language (XPath) Version 1.0 (W3C Recommendation 16 November 1999) at www.w3.org/TR/xpath.

13. Appendix A – Inventory XML

Sample XML File for XPath Syntax (inventory.xml)

This XML file is used by some topics in the XPath Reference documentation. This file represents a fragment of a bookstore inventory database.

XML File (inventory.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="myfile.xsl" ?>
<bookstore specialty="novel">
 <book style="autobiography">
  <author>
   <first-name>Joe</first-name>
   <last-name>Bob</last-name>
   <award>Trenton Literary Review Honorable Mention</award>
  </author>
  <price>12</price>
 </book>
 <book style="textbook">
  <author>
   <first-name>Mary</first-name>
   <last-name>Bob</last-name>
   <publication>Selected Short Stories of
    <first-name>Mary</first-name>
    <last-name>Bob</last-name>
   </publication>
  </author>
  <editor>
   <first-name>Britney</first-name>
   <last-name>Bob</last-name>
  </editor>
  <price>55</price>
 </book>

 <magazine style="glossy" frequency="monthly">
  <price>2.50</price>
  <subscription price="24" per="year"/>
 </magazine>
 <book style="novel" id="myfave">
  <author>
   <first-name>Toni</first-name>
   <last-name>Bob</last-name>
   <degree from="Trenton U">B.A.</degree>
```

```
    <degree from="Harvard">Ph.D.</degree>
    <award>Pulitzer</award>
    <publication>Still in Trenton</publication>
    <publication>Trenton Forever</publication>
   </author>
   <price intl="Canada" exchange="0.7">6.50</price>
   <excerpt>
    <p>It was a dark and stormy night.</p>
    <p>But then all nights in Trenton seem dark and
    stormy to someone who has gone through what
    <emph>I</emph> have.</p>
    <definition-list>
     <term>Trenton</term>
     <definition>misery</definition>
    </definition-list>
   </excerpt>
  </book>

  <my:book xmlns:my="uri:mynamespace" style="leather" price="29.50">
   <my:title>Who's Who in Trenton</my:title>
   <my:author>Robert Bob</my:author>
  </my:book>
</bookstore>
```