

Simple Expressions

\$VarName
(Expr)
()
• (one dot: self)
QName (Expr , ...)
QName ()
IntegerLiteral
DecimalLiteral
DoubleLiteral
StringLiteral

Arithmetic Expressions

+ Expr Expr + Expr
- Expr Expr - Expr
Expr * Expr Expr **div** Expr
Expr **idiv** Expr Expr **mod** Expr

Creating Sequences

Create a sequence from a list of items:

Expr , ...

Note: A sequence list must usually be parenthesized.

Repeat over one or more sequences, returning a sequence of results:

for VariableBinding , ... **return** Expr

where a VariableBinding is:

\$VarName in Expr

Create a numeric sequences, from lower bound to upper bound:

Expr **to** Expr

All the items appearing in either sequence:

Expr **union** Expr
Expr | Expr

Only items appearing in both sequences:

Expr **intersect** Expr

All items in the first sequence not in second:

Expr **except** Expr

Comments in XPath Expressions

(: This is a comment within an XPath expr :)

Testing

Test if the condition is satisfied for at least one combination of the bound expressions:

some VariableBinding , ... **satisfies** Expr

Test if the condition is satisfied for all of the bound expressions:

every VariableBinding , ... **satisfies** Expr

Select one or the other of two possibilities:

if (Expr) **then** Expr **else** Expr

Either or both of two tests:

Expr **or** Expr Expr **and** Expr

Test if they are the same node:

Expr **is** Expr

Test if a node appears before or after another:

Expr << Expr Expr >> Expr

Test an expression's dynamic type:

Expr **instance of** SequenceType

Test if an expression can be converted to a type:

Expr **castable as** AtomicType

Expr **castable as** AtomicType?

Compare two atomic values:

Expr **eq** Expr Expr **ne** Expr

Expr **lt** Expr Expr **le** Expr

Expr **gt** Expr Expr **ge** Expr

Compare all items in one sequence to all items in a second, and return if true for any pair of values:

Expr = Expr Expr != Expr

Expr < Expr Expr <= Expr

Expr > Expr Expr >= Expr

Type Modification Expressions

Use as without converting:

Expr **treat as** SequenceType

Use as, converting as needed and doable:

Expr **cast as** AtomicType

Expr **cast as** AtomicType?

XPath 2.0:

<http://www.w3.org/TR/xpath20/>

XSL-List:

<http://www.mulberrytech.com/xsl/xsl-list>

Path Expressions

/ Top level, document root
/ Step At top level
Step Relative to current node
// Step Anywhere within document
Path / Step Immediately within Path
Path // Step Anywhere within Path

Where a Step is one of:

Expr
AxisName::NameTest
AxisName::KindTest
@NameTest (attribute test)
NameTest (child element test)
KindTest (child node test)
.. (two dots: parent test)

Followed by zero or more predicates:

[Expr]

Where an AxisName is one of:

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

Where a NameTest is one of:

QName
*
NCName:*
*:NCName

Where a KindTest is one of:

attribute (AttributeName)
attribute (AttributeName , TypeName)
attribute (*)
attribute (* , TypeName)
attribute ()
comment ()
document-node (element ...)
document-node (schema-element ...)
document-node ()
element (ElementName)
element (ElementName , TypeName)
element (*)
element (* , TypeName)
element ()

node ()

processing-instruction (NCName)

processing-instruction (StringLiteral)

processing-instruction ()

schema-attribute (AttributeName)

schema-element (ElementName)

text ()

Names and Types

XML QNames, with or without a colon-separated prefix, is use for all of:

VarName
AttributeName
ElementName
TypeName
AtomicType

A SequenceType is one of:

empty-sequence ()

KindTest

item ()

AtomicType

Where KindTest, **item()** or AtomicType can be optionally followed by:

? (may be empty sequence)\
+ (is a non-empty sequence of the type)
* (is a sequence of the type, empty or not)

Operator Precedence:

- 1 , (comma)
- 2 **for some every if**
- 3 **or**
- 4 **and**
- 5 = != < <= > >=
- 6 **to**
- 7 (two-argument) + -
- 8 * **div idiv mod**
- 9 **union |**
- 10 **intersect except**
- 11 **instance of**
- 12 **treat as**
- 13 **castable as**
- 14 **cast as**
- 15 (one-argument) + -
- 16 / //
- 17 step node-test \$name
(Expr) function-call literal

Relative Location Paths

Relative Location Paths traverse the document from the context node

para

para element children
Also – **child::para**

@type

the **type** attribute
Also – **attribute::type**

../title

the **title** element children of the parent

* except title

child elements except **title** elements
Also – ***[not(self::title)]** (works in XPath 1.0)

ancestor::sec

all **sec** ancestor elements

ancestor::sec/@n

all **n** attributes on **sec** ancestor elements

list/(item | step)

item and **step** element children of **list** children, in document order

list/item, list/step

item element children of **list** children followed by **step** children of **list** children

preceding-sibling::step

all preceding sibling **step** elements

preceding-sibling::*[1][self::step]

the directly preceding sibling element, if it is a **step** (otherwise nothing)

descendant::div[last()]

the last **div** descendant of the current node

../div[last()]

div descendants that are the last child **div** of each of their parents

preceding::pb[1]

the first (most immediate) preceding **pb**

ancestor::sec//pb intersect preceding::pb

pb elements inside the same **sec** element as the context node, preceding it

p[normalize-space()]

p child elements that have a non-whitespace value (text content)

*[not(node())]

empty element children (i.e., element children with no node children)

*[not(node() except (comment()|processing-instruction()))]

element children that are empty (have no children) except for comments or processing instructions

step[position() gt 1]

all **step** element children but the first

step except *[1]

step element children but the first

step[position() le 4]

the first four **step** element children
Also – **step[position() = (1 to 4)]**

step[position() mod 2]

odd-numbered **step** children

step[not(position() mod 2)]

even-numbered **step** children

*[position() le 4] intersect step

from the first four element children, the **step** children

ancestor-or-self::*[exists(@lang)][1]/@lang

the closest **lang** attribute on the context node or an ancestor element

Expressions that are not Location Paths

(@class,'none')[1]

the **class** attribute, or if it does not exist, the string "**none**".

Also –

if (exists(@class)) then @class else "none"

//*[name()]

the names of all elements, in document order

distinct-values(//*[name()])

the names of all elements, in document order, with duplicates removed

//name/string-join((first, last), '')

a sequence of strings constructed from the **name** elements in the document, each one concatenating the values of its **first** and **last** element children, in that order, joining them with spaces

Also – **for \$n in //name return string-join((\$n/first,\$n/last),'')**

//*[count(ancestor-or-self::*)]

a sequence of numbers representing the depth of each element in the document

max(//*[count(ancestor-or-self::*)])

the maximum depth of all elements in the document (a number in a singleton sequence)

for \$stooge in ('Moe','Larry','Curly')

returncount(//p[contains(.,\$stooge)])
the counts of all **p** elements in the document mentioning each of "Moe", "Larry" and "Curly", in that order

index-of(('Moe','Larry','Curly'), speaker[1])

if the first **speaker** element child has the value "Moe", then 1; if "Larry", then 2; if "Curly", then 3; otherwise the empty sequence (i.e., no value)

(: You've got to be kidding me. :)

do nothing. A comment is just a comment.

2008-07-21

XPath 2.0 Quick Reference

See also the "XQuery 1.0 & XPath 2.0 Functions & Operators Quick Reference"

Sam Wilmott
sam@wilmott.ca
<http://www.wilmott.ca>

and

Mulberry Technologies, Inc.
17 West Jefferson Street, Suite 207
Rockville, MD 20850 USA
Phone: +1 301/315-9631
Fax: +1 301/315-8285
info@mulberrytech.com
<http://www.mulberrytech.com>



© 2007-2008 Sam Wilmott and
Mulberry Technologies, Inc.

Absolute Location Paths

Absolute Location Paths traverse the document starting at the top (the root), and can be recognized by their initial / (forwardslash).

/book/bookinfo/abstract

an **abstract** element child of a **bookinfo** child of the **book** document element

Also –

/child::book/child::bookinfo/child::abstract

//para

all **para** elements in the document

Also – **/descendant-or-self::*/child::para**

Also – **/descendant::para**

/descendant::para[1]

the first **para** element in the document

Also – **(//para)[1]**

//@order-by

all **order-by** attributes in the document

//list[exists(ancestor::list)]

all **list** elements that have ancestor **list** elements

//list[not(ancestor::list)]

all **list** elements that do not have ancestor **list** elements

Also – **//list[not(exists(ancestor::list))]**

Also – **//list[empty(ancestor::list)]**

//(* except title)

all elements except **title** elements

Also – **//*[not(self::title)]** (works in XPath 1.0)

//processing-instruction()[not(ancestor::sec/@n = 1)]

all processing instructions with no **sec** ancestor elements with **n** attributes equal to 1

//para[matches(.,'[X|x]{3}')

all **para** elements whose value includes the regular expression **[X|x]{3}**

Tip – **[X|x]{3}** matches three **X** or **x** characters appearing in a row

//sec[@id = //@rid/tokenize(.,'\s+')

all **sec** elements with **id** attributes whose values are also given as a value by a tokenized **rid** attribute anywhere in the document

Also – **//sec[@id = \$rid-values]** where **\$rid-values** is

distinct-values(//@rid/tokenize(.,'\s+'))
Tip – use

distinct-values(//@rid/tokenize(.,'\s+')) to remove duplicates from the list of tokenized **@rid** values

Tip – the regular expression **\s+** matches any contiguous sequence of spaces (space, linefeed or tab characters)