COIN-078B
Internet Programming with XML

XPath - Patterns, Expressions and Functions

Module Objectives

You'll learn...

- what XPath is and its role on XML & XSLT
- the differences between patterns, expressions and functions
- how to write patterns and expressions using XPath syntax
- how to use XPath functions to test conditions
- how to use XPath patterns, expressions and functions to select specific data

Reading Assignments:

- **Textbook:** Chapter 9

Introducing XPath

The XPath language is the product of cooperation between the XSL and XPointer Working Groups. As XSLT, the XSL Transformation Language, and XPointer were being developed, it became clear that they had similar needs. In order to operate, both specifications need a way to point to a specific thing or a set of things in an XML document.

It would have been counter-productive to create two very similar but different languages, so the WGs agreed to harmonize. XPath is the result of that harmonization. However, the needs of XSLT and XPointer are not absolutely identical, so XPath provides only core functionality. XSLT and XPointer both make application-specific additions to it. Future standards, for example a query language standard, are likely to start with XPath as a foundation and add their own extensions, too.

XPath is a language that describes how to locate specific elements (and attributes, processing instructions, etc.) in a document. XPath uses a compact, string-based syntax, rather than a structural XML-element based syntax. This allows XPath expressions to be used both in XML attributes and in URIs.

In a nutshell, XPath expressions allow you to address into an XML document. The heart of this addressing mechanism is the *location path*. A location path describes how something (or things) may be found. By way of very rough analogy, you can think of a location path as being similar to a set of street directions. If I wanted to describe where a particular store was, I might say "it's the second store on the left after the third light." Similarly, I might describe a figure in an XML document as "the second figure in the third section."

An important aspect of both of these paths is that they are relative to the starting location. In XPath this is the concept of the "*context node*". The context node is where the addressing starts. XSLT and XPointer have different ways of determining what the context node is, but for the time being don't worry too much about that. Just be aware that a location path is really a description of how to get from where you are to somewhere else.

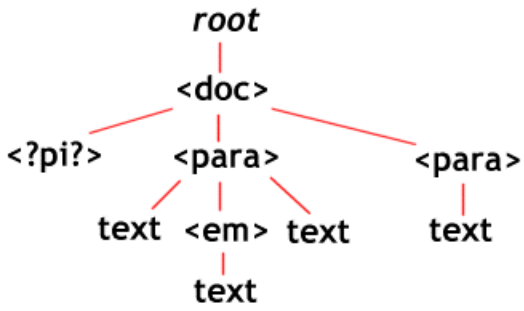
XPath Data Model

Before we look more closely at location paths, let's take a moment to consider the data model used by XPath. XPath treats the XML document as a logical tree, independent of its physical representation. In this tree, there are nodes for each type of construct that can appear in an XML document. In all there are seven node types: *element*, *attribute*, *text*, *processing instruction*, *comment*, *namespace*, and *root*. The first five of these represent what you would expect: the elements, attributes, text (character data), processing instructions, and comments in a document. Namespace nodes represent the namespaces used in the document, as defined by *XML Names*. The root node is the root of the document. In every XML document, there is exactly one root node, which the node that contains the (optional) prologue and the document element.

For example, this simple document:

```
<doc>
<?pub caret?>
<para>Some <em>emphasis</em> here.</para>
<para>Some more stuff.</para>
</doc>
```

Might be represented as follows:



This tree representation makes clear the relationships between the various nodes. For example, the children of doc are clearly a processing-instruction node and two para element nodes. Four nodes comprise the descendants of the first para node: two text nodes and an em node are children, and a text node is a grandchild (the child of em is a descendant of the para).

As we shall see, location paths describe the address of one node with respect to another using these relationships.

I'd like to draw your attention to one last aspect of this tree structure, namely that it is ordered. The diagram above reads from top to bottom and left to right. The processing instruction node comes before the paragraph node. Likewise, the em element node comes after the text node that logically precedes it. It is possible to navigate in XPath by counting nodes and selecting them by ordinal position with respect to document (or reverse document) order. For example, I could address the second para child of doc.

Location Paths

Now it's time for a more technical inspection of location paths. A *location path* describes a path from one point in an XML document to another, providing a way for the items in an XML document to be addressed. XPath location paths resemble filesystem paths (/html/head/title). A *location path* is composed of a series of *steps*. Each *step* consists of a *basis* and optional *predicates* . A *basis* is an *axis name* and a *node test*.

Like many specifications, it looks more complex than it really is because each component of the language has to have a precise technical name in order for the specification to describe it. Let's start at the end and work our way backward.

A *node test* identifies a type of node in the document. Usually, it's the name of an element, but it may also be a function that identifies a node type, such as `text()` for text nodes or simply `node()` for any node type. Now consider a node test by itself. If I simply said, find a para element, that wouldn't be very precise. You could find the child para elements, or the preceding paras, or the next, or even ancestor paras. You could even find the context node itself, if you happen to be in a para to begin with. The *axis name* distinguishes between these tests: `child::para` finds the para elements that are children of the context node while `preceding-sibling::para` finds the preceding para elements that are siblings to the context node.

There are 13 axes defined in XPath:

ancestor, ancestor-or-self

> Locate the specified node-test among the ancestors of the context node. If ancestor-or-self is used, the context node is considered one of the ancestors. This distinction may seem a bit odd, but it is very useful in some contexts, for example searching for an attribute that may be specified on the context node or may be inherited from one of its ancestors.

attribute

Locate the specified node-test among the attributes of the context node. (abbreviated @)

child

Find the node-test among the children of the context node. (the default axis)

descendant, descendant-or-self

Locate the specified node-test among the descendants of the context node. The descendants of a node are all of its children, all of its children's children, etc. (abbreviated //)

following

Find the node-test among the following elements. The following elements are all of the elements that come after the context node, excluding its descendants. In other words, all the elements whose start tag comes after the end tag of the context node, in document order.

following-sibling

Locate the specified node-test among the following siblings of the context node.

namespace

The namespace axis contains all of the namespaces that are open at the context node. This axis is empty if the context-node is not an element.

parent

The parent axis refers to the parent of the context node. (abbreviated ..)

preceding

Find the node-test among the preceding elements. The preceding elements are all those elements that come before the context node, excluding its ancestors.(returned in reverse-document order)

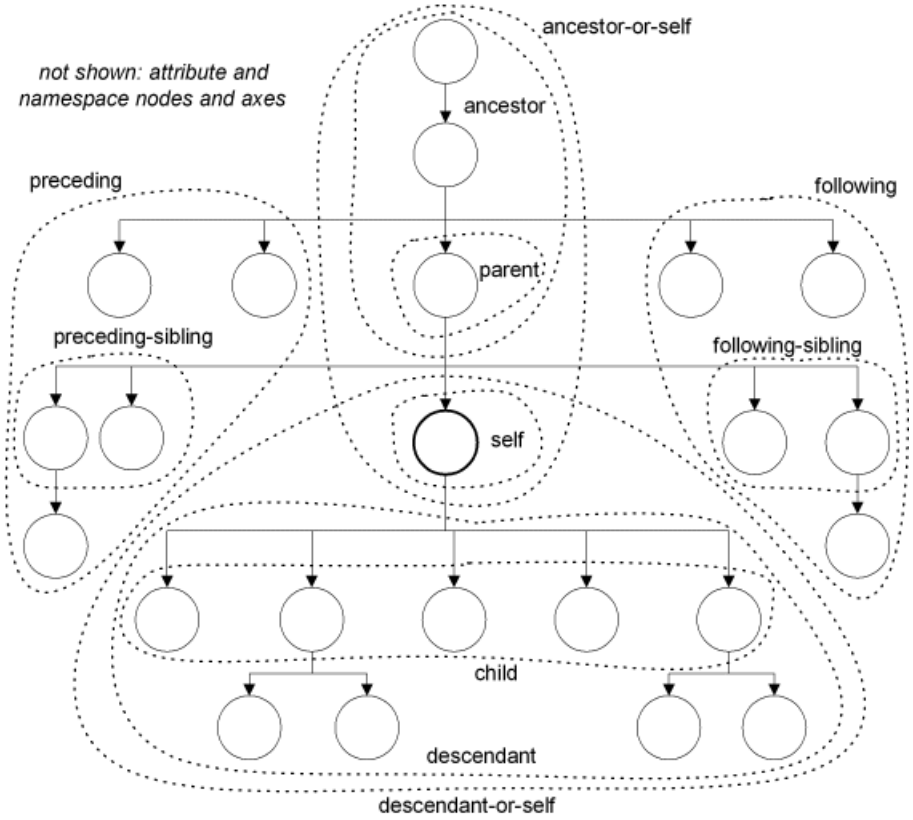
preceding-sibling

Locates the specified node-test among the preceding siblings of the context node. (returned in reverse-document order)

self

The self axis selects the context node. (abbreviated .)

The following graphic, courtesy of Crane Softwrights, demonstrates the axes graphically:



An *axis* and a *node-test* form a *basis*. That is, the node-test gives you your general target, and the axis gives you where to look, so this gives you a 'basis' on which to start out. But it may be necessary to get still more specific. A basis may be further qualified with predicates. Predicates are filters that restrict the nodes selected by an XPath expression. A *predicate* occurs in square brackets [] after the basis. For example, `child::para` is a basis that selects all of the para children of the context node. Adding the predicate `[3]: child::para[3]` selects only the third paragraph. `/doc/para[position() mod 2 = 0]` selects even ones

A *basis* with *predicates* forms a *step*. Steps may be joined with slashes to form a complete *location path*. The slashes in a location path serve much the same function as slashes in a directory path (from which location paths derive their name). Each slash moves "down" the tree from the preceding step. So, for example, `child::section` finds the section children and `child::section/child::para` finds all the para children of all the section children.

A Note on Document Order and Position

The meaning of a predicate that tests order, such as `para[3]`, depends on the axis that is used. Most axes return nodes in document order, but there are exceptions: the ancestor and preceding axes use reverse document order and the attribute and namespace axes are unordered. In other words, `ancestor::section[1]` returns the nearest section ancestor, the one closest in reverse document order.

This is a somewhat subtle point, but it's worth remembering. Be aware also of the following "gotcha", if you use parentheses for grouping, you will always get forward document order. This means that, somewhat counter-intuitively, `ancestor::section[1]` and `(ancestor::section)[1]` may return different nodes. The latter returns the first (in document order) of all your section ancestors, in other words, the one farthest away in reverse document order.

This is someone inconvenient, but it isn't a real limitation, you can always use the last() functions to address from the tail of the list instead of the head.

Abbreviating Location Paths

All this business of typing `axis::node-test/axis::node-test` is nice and unambiguous, but it's also a bit tedious. It would be nice if the common cases could be made a little simpler. And, indeed, they can. XPath defines both a flexible, extensible verbose syntax and an abbreviated syntax for the common cases.

The abbreviations can be summarized as follows:

1. The *child::* axis is the default axis for an unqualified node test. (child::body is the same as body)
2. The @ character is an abbreviation for the attribute:: axis.
3. A * matches all of the element children of the context node, @* matches all the attributes of the context node.
4. The . character is an abbreviation for self::node().
5. The string .. is an abbreviation for parent::node().
6. The string // is an abbreviation for descendant-or-self::node().

These abbreviations combine to make many location paths much simpler and are consistent with what your intuition might expect. For example: `self::node()/descendant-or-self::node()/child::para` becomes simply `.//para` and selects all of the para elements among the descendants of the context node.

Locating the Root Element

The only absolute location that you can get to from anywhere in an XML document is the root node. If a location path begins with a slash, it is considered relative to the root node. So, while section finds all of the section children of the context node, `/section` finds all the section children of the root node (which will find either exactly one node, if the document element is section, or none).

A path that begins with `//`, finds descendants of the root node, so `//para` finds all of the paragraphs anywhere in the document.

Examples

A few examples ought to make things a little clearer.

`para`

The simplest form of address is simply the name of an element. This path locates all of the para children of the context node.

`@role`

Using an @-sign matches an attribute. This path locates the role attribute of the context node.

`list/item`

This path locates all of the item children of all of the list children of the current node.

`list//para`

This path finds all of the para elements (nested arbitrarily deep) under all of the list children of the context node. In other words, this location path matches `list/item/para`, `list/item/note/para` , and every other combination of tags that amounts to a para somewhere under a list.

`note[@class="warning"]`

This location path finds only note children that have a class attribute with the value "warning".

`section[title="Introduction"]`

Finds the section(s) with a title of "Introduction".

`section[3]`

Finds only the third section. This is actually a shortcut for `section[position()=3]`. In general, you have to use the longer form, the shortcut only works for literal, decimal indexes.

```
//section[title="Examples"]/example[last()]
```

Finds the last example in all sections (anywhere in the document) that have a title of "Examples".

XPath Functions

In the preceding sections, I have been rather cavalier about the expressions that I placed in predicates. I should explain that in addition to the semantics for location paths, XPath includes a small expression language supporting basic mathematics and an extensible set of functions. These functions allow you to easily construct quite complex location paths. For example, if you wanted to select all of the odd-numbered elements of a list, you could do so with the following location path:

```
list/item[position() mod 2 = 1]
```

The position() function returns the position of an element among its siblings and mod returns the remainder after division, so this predicate selects only the first, third, fifth, etc. item children of list.

XPath defines a core set of functions and operators. The list of supported functions may be extended by the specifications that use XPath.

Resources

- XML.com: XML Path Language (XPath)
- XML in a Nutshell: XPath
- XPath Tutorial
- W3Schools: XPath Tutorial
- XML for the World Wide Web: Chapter 11 - XPath: Patterns and Expressions
- W3C : XML Path Language (XPath) Version 1.0
- XSLT and XPath Quick Reference
- MSDN XPath Reference
- XSL Concepts and Practical Use
- XPath Axis-Name CheatSheet
- XML for Data: Styling with schemas
- XSLT and XPath Sample Viewer (IE only)
- Saxon XSLT Processor

Hands-on Assignment:

- **Assignment 1** due **Week 3**