Get started with XPath

Learn the basics

Skill Level: Introductory

Bertrand Portier
Software Engineer
EMC

11 May 2004

This tutorial introduces and covers most aspects of the XML Path Language, or XPath. It is aimed at people who do not know XPath or who want a refresher. If you plan to use XSLT, you should take this tutorial first. You will learn: what XPath is; the syntax and semantics of the XPath language; how to use XPath location paths; how to use XPath expressions; how to use XPath functions; and how XPath relates to XSLT.

Section 1. Tutorial introduction

Should I take this tutorial?

This tutorial introduces and covers most aspects of the XML Path Language, or XPath. It is aimed at people who do not know XPath or want a refresher. If you plan to use XSLT, you should take this tutorial first. You will learn:

- What XPath is
- The syntax and semantics of the XPath language
- How to use XPath location paths
- How to use XPath expressions

- How to use XPath functions
- How XPath relates to XSLT

XPath is a W3C standard. This tutorial focuses on version 1.0. Refer to Tutorial wrap-up for information on the upcoming XPath 2.0.

Prerequisites

This tutorial assumes basic knowledge of XML. For example, you should know what elements, attributes, and values are. If you aren't familiar with these, I recommend you first take the "Introduction to XML" tutorial, listed in Resources.

You will also need a text editor and a Web browser. Although it's not a prerequisite, you may also find it useful to have an XML editor, as it will include an XPath evaluator that allows you to test the XPath expressions used here.

XML-related technologies

Listed below are the technologies mentioned in this tutorial. Skip this if you are already familiar with XML-related technologies. All of these technologies are W3C specifications.

- **XML** is the eXtensible Markup Language. It is the basis for the other technologies listed below (and many more), and is very widely used in the industry because of its extensibility.
- **XML Schema**, sometimes referred to as XSD, defines the rules for the data contained in XML documents.
- **XSL** (eXtensible Stylesheet Language) and **XSLT** (XSL transformations) are used to present XML documents in a different format -- for example, HTML.
- **XPath** -- hmm... that's a good question. ;-)

Section 2. About the example used in this tutorial

Overview

In this tutorial, you will learn XPath by writing the presentation layer of an auction site application. You will specify the XPath expressions inside an XSLT stylesheet that's used to present XML documents containing auction items. All the files used in this tutorial are in the zip file, x-xpathTutorial.zip (see Resources for a link), including:

- **XPath/AuctionItemList.xsd** - an XML Schema document that defines the data format of the auction items.
- **XPath/AuctionItemList.xml** - an XML file that contains a list of auction items; it is the data for the example.
- **XPath/AuctionItemSummary-Base.xsl** - an XSLT stylesheet that defines what a Web browser will display when it loads AuctionItemList.xml; it contains the data's presentation rules.
- **XPath/AuctionItemSummary-Section5.xsl** - the solution in Location paths .
- **XPath/AuctionItemSummary-Section6.xsl** - the solution in Expressions .
- **XPath/AuctionItemSummary-Section7.xsl** - the solution in Function library .

AuctionItemList.xsd

AuctionItemList.xsd contains the business rules for the auction item and auction items list data, described in XML Schema language:

An auction item list has only one root element called `list` of type `auctionItemList`.

An `auctionItemList` is composed of zero or more `item` elements of type `auctionItem`.

An `auctionItem` is composed of five elements (`bidIncrement`, `currentPrice` of type `price`, `endOfAuction`, `description`, and `sellerId`) and one attribute group of type `itemAttributes`.

A `price` is a positive decimal value with two decimal places and must have a `currency` attribute of type `customCurrency` associated with it.

A `customCurrency` must be one of `USD`, `GBP`, or `EUR`.

An `itemAttributes` group must contain one string attribute `type`, one string attribute `id`, and one boolean attribute `private`, that is `false` by default.

A `type` attribute must be one of the following: `Unknown`, `Traditional`, `BidOnly`, `FixedPrice`, or `IndividualOffer`.

If you want to learn more about XML Schema, see Resources for more *developerWorks* articles and tutorials.

AuctionItemList.xml

AuctionItemList.xml conforms to the XML schema defined in AuctionItemList.xsd and contains a list of type `auctionItemList`. This list contains seven items. The `xsi:schemaLocation` attribute of the list root element indicates that this XML document must conform to the AuctionItemList.xsd schema.

That takes care of the data format, but what about the presentation? How do you specify which XSLT stylesheet to use to display this XML document in a Web browser? This is defined in the second line of the XML document:

```
<?xml-stylesheet type="text/xsl" href="AuctionItemSummary-Base.xsl"?>
```

Here, I state that the AuctionItemSummary-Base.xsl stylesheet should be used. The data itself has been chosen so that the use of XPath can be demonstrated to show specific data properties. When no XML stylesheet document is linked to AuctionItemList.xml, then a Web browser simply shows the XML contents and it looks like the following:

Figure 1. AuctionItemList.xml

AuctionItemSummary-Base.xsl

AuctionItemSummary-Base.xsl is the XSLT stylesheet that defines the rules used by an XSLT processor to display `AuctionItemList` XML documents. It uses XPath expressions to find information in the XML document and display it in an HTML table. I will focus in more detail on the use of XPath in XSLT in XPath overview . Here, I describe briefly the contents of AuctionItemSummary-Base.xsl. It defines templates that are activated when XML documents are processed. Which template is activated depends on the XPath expression documented in the `match` attribute of this `template` element. For example, the following snippets, taken from AuctionItemSummary-Base.xsl, are XPath expressions:

- `"/"`
- `"list"`
- `"item"`

The information that displays when a template is activated is defined by its `value-of` elements' `select` attributes. These attributes' values are also XPath expressions. For example:

- `"description"`
- `"currentPrice"`
- `"@type"`
- `"@private"`

In each section (Location paths , (Expressions , and (Function library), you will modify AuctionItemSummary-Base.xsl to display the information in a different way.

By this point, you should have looked at the files in a text/XML editor. Now you can open AuctionItemList.xml in your favorite Web browser to see the output generated by an XSLT processor based on the stylesheet. You should see something similar to the following:

Figure 2. The base auction item table

Section 3. XPath overview

What is XPath?

The **XML Path Language (XPath)** is a set of syntax and semantics for referring to portions of XML documents. XPath is intended to be used by other specifications such as XSL Transformations (XSLT) and the XML Pointer Language (XPointer). To help you understand what XPath is, I will start by showing examples related to AuctionItemList.xml.

XPath expressions identify a set of **nodes** in an XML document. This set of nodes can contain zero or more nodes. For example, the XPath expression `/list`, if applied to AuctionItemList.xml, identifies one single node -- the `list` root element.

The XPath expression `/list/item` identifies all the `item` elements.

XPath uses a notation with forward slashes (`/`) similar the UNIX shell. This is so that XPath can be used in Uniformed Resource Identifiers (URIs), such as URLs. This is actually where XPath's name comes from: the use of a path notation as in URLs.

Legal XPath expressions can include **predicates**. Predicates contain boolean

expressions, which are tested for each node in the context node-set. If true, the node is kept in the set of nodes identified; otherwise, the node is discarded. Predicates are useful in reducing the result set. For example, the following XPath expression identifies the second item only:

```
/list/item[currentPrice>1000.0]
```

XPath expressions can refer to attributes as well as elements in an XML document. When referring to an attribute, the @ character is used. For example, the following XPath expression identifies `currentPrice` elements whose currency attributes contain the value `EUR`:

```
/list/item/currentPrice[@currency="EUR"]
```

XPath also provides **functions** , which can come in very handy. I'll show you these in more detail in Function library , but here is a taste of it. The XPath expression below identifies the `description` element of the item whose `type` attribute is `"IndividualOffer"` (and has the value `2MP digital camera`):

```
/list/item[starts-with(@type,"Ind")]/description
```

Test the above XPath expressions in your favorite XML editor: Open AuctionItemList.xml and enter the expressions in the XPath evaluator to see which nodes are selected.

That's it -- you've now been introduced to XPath! So far, you've learned that XPath is a language for identifying parts of XML documents. You've seen what an XPath expression looks like and how it refers to elements and attributes inside XML documents. I've also shown you how XPath provides functions for manipulating data. However, this is just a quick overview; I will discuss all these points in more detail -- as well as more aspects of XPath -- in the remaining sections. For example, I'll examine XPath namespaces and special characters (such as `//` and `*`) and show you that not all XPath expressions have the form shown in the examples above (called abbreviated location paths).

XSLT, XLink, and XPointer

XSLT, XLink, and XPointer are all W3C standards. XSLT and XPath, along with XSL Formatting Object (XSL-FO), form the W3C eXtensible Stylesheet Language (XSL) family of specifications. (see Resources if you want to look at these specifications.)

Presenting : XSLT uses XPath extensively for matching -- that is, testing whether a node matches a given pattern. XSLT specifies the context used by XPath. You should understand XPath if you want to work with XSLT. In About the example used in this tutorial , you saw that the AuctionItemSummary-Base.xsl stylesheet contains

XPath expressions. These XPath expressions are used by XSLT to find elements that match criteria in the source document, and also to display information in the result document. XSLT also makes use of XPath functions to perform arithmetic or string manipulation operations.

Linking: XLink provides a generalization of the HTML hyperlink concept in XML. XLink defines a syntax for elements to be inserted into XML documents in order to link resources together and to describe their relationship. These links can be unidirectional, like HTML's hyperlinks, or more complex. XLink uses XPointer to locate resources.

Pointing: XPointer is an extension to XPath that provides addressing into XML documents and their internals. XPointer generalizes the notion of XPath nodes with the concept of XPointer locations, points, and ranges. XPointer also specifies the context used during XPath evaluation and provides extra functions that aren't available in XPath.

XPath is essential to the specifications mentioned above. In fact, the XPath specification explicitly states that XPath is designed to be used by XSLT and XPointer.

Here is a recap of the XML technologies I have talked about so far:

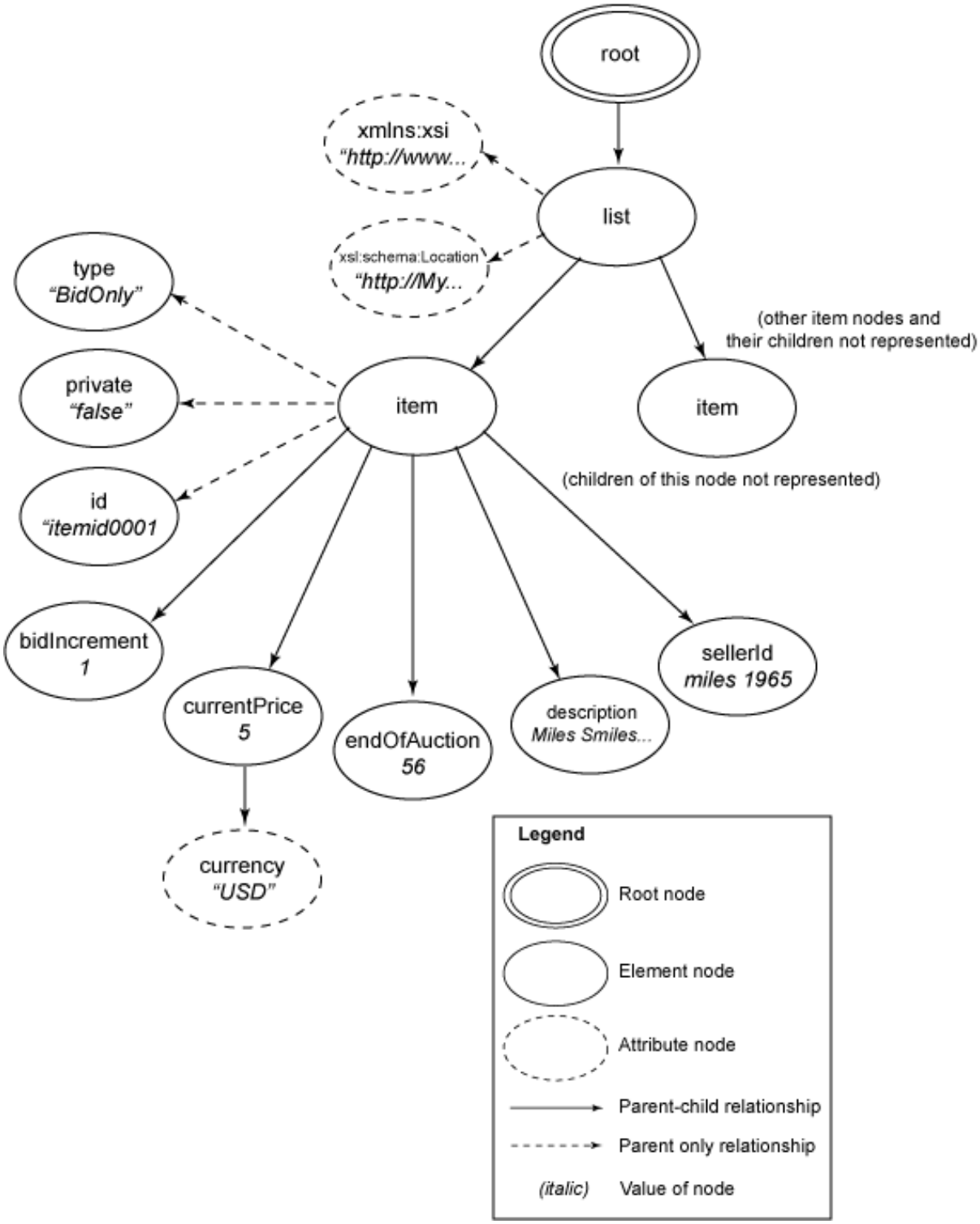
- **XML**: basis for other technologies (data)
- **XML Schema**: data format rules
- **XSLT**: data presentation/matching
- **XLink**: linking
- **XPointer** and **XPath**: addressing

Section 4. XPath terminology

What is an XPath node?

XPath sees an XML document as a tree of **nodes**. Nodes can be of different types, such as **element** nodes or **attribute** nodes. Some types of nodes have names that consist of a nullable XML namespace URI and a local part. For example, the figure below shows the XPath representation of AuctionItemList.xml as a tree of nodes:

Figure 3. XPath's view of AuctionItemList.xml



One special type of node is the **root** node. An XML document contains only one such node and it is the root of the tree, containing the whole of the XML document. Note that the root node contains the root element as well as any processing,

declaration, or comment nodes that appear before or after the root element. In the example, the children of the root node are:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?xml-stylesheet type="text/xsl" href="AuctionItemSummary-Base.xsl"?>
```

and

```
<list ...>
  <item ...>
    ...
  </item>
  <item ...>
    ...
  </item>
  ...
</list>
```

No node type exists for XML declarations (such as `<?xml version="1.0" encoding="UTF-8"?>`) or Document Type Definitions (DTDs). It is therefore not possible to refer to such entities in XPath.

Element nodes represent every element in an XML document. Attribute nodes belong to element nodes and represent attributes in the XML document. However, attributes that start with `xmlns:` are represented in XPath with namespace nodes. Other types of nodes include text nodes, processing instruction nodes, and comment nodes.

Out of context?

XPath evaluates expressions relative to a **context**. This context is usually specified by the technologies that extend XPath, such as XSLT and XPointer. An XPath context includes a context node, context size, context position, and other context data. From a context standpoint, the context node is of most interest here. When the context node is the root node, `list/item` refers to the seven `item` elements in AuctionItemList.xml. When the context node is another node -- for example, the first `item` element -- `list/item` does not refer to anything in the XML document.

In XSLT, the values of `select` attributes are XPath expressions. For example, in AuctionItemSummary-Base.xsl, the `xsl:apply-templates` and `xsl:value-of` elements have `select` attributes whose values (XPath expressions) are, among others, `list`, `item`, or `itemId`. *In XSLT, the context node is the current node being evaluated*. XSLT templates can be activated several times for an XML document and produce different results. With AuctionItemList.xml, the first and second templates (`match="/"` and `match ="list"`, respectively) are activated once, and the third template (`match="item"`) is activated seven times (once for each `item`

element). The first time the "item" template is activated, the context node is the first `item` element in the XML document ("Miles Smiles album, CD") and, for example, the value of `<xsl:value-of select="@id"/>` is `itemId0001`. The second time the XSLT template is activated, the context node is the second `item` element (the "Coltrane Blue Train" CD) and the value of `<xsl:value-of select="@id"/>` is `itemId0002`. Note that had I used `/list/item/@id` as opposed to `@id` for the `select` attribute, the value of the `xsl-value-of` element would have been null.

Section 5. Location paths

What is a location path?

Location paths are the most useful and widely used feature of XPath. A location path is a specialization of an XPath expression (described in Expressions). A location path identifies a set of XPath nodes relative to its context. XPath defines two syntaxes: the **abbreviated** syntax and the **unabbreviated** syntax.

In this tutorial, I only talk about the abbreviated syntax because it is the most widely used; the unabbreviated syntax is also more complex. If you are interested in the unabbreviated syntax, have a look at the XPath 1.0 specification (see Resources).

The two types of location paths are **relative** and **absolute**.

A **relative** location path is a sequence of location steps separated by `/`. For example:

```
list/item[currentPrice<20.0]
```

This location path consists of two location steps: the first, `list`, selects a set of nodes relative to the context node; the second, `item[currentPrice<20.0]`, selects a set of nodes in the subset identified by the first step; and so on, if there are more nodes.

An **absolute** location path consists of a `/`, optionally followed by a relative location path, with `/` referring to the root node. An absolute location path is basically a relative location path evaluated in the context of the root node, for example:

```
/list/item[currentPrice<20.0]
```

With absolute location paths (location paths that start with `/`), the context node isn't

meaningful because the path is always evaluated from the root node.

Useful syntax

The abbreviated syntax provides a set of useful characters (most of which you saw in XPath overview). I will now enumerate the most commonly used characters and give examples relative to the root node of AuctionItemList.xml -- that is, with the context node being the root node of AuctionItemList.xml.

@ is used to refer to attributes. For example, the location path `@currency` identifies the `currency` attribute. `list/item[@private]` identifies the `item` elements with a `private` attribute -- meaning, all the `item` elements in AuctionItemList.xml.

***** is used to refer to all the elements that are children of the context node. **@*** is used to refer to all the attributes of the context node.

[] can also be used to refer to specific elements in an ordered sequence. For example, `list/item[2]` refers to the second `item` element. This is actually a predicate (described next in Predicates).

// is used to refer to all children of the context node. For example, `//item` refers to all `item` elements, and `//list/item` refers to all `item` elements that have a `list` parent (that is, all the `item` elements in the example).

. is used to refer to the context node itself. For example, `.` selects the context node, and `.//item` refers to all the `item` elements that are children of the context node.

.. is used to refer to the parent of the context node. For example, `../item` would refer to the first `item` element in the context of the first `bidIncrement` element.

Predicates

Predicates are used in location paths to filter the current set of nodes. A predicate contains a **boolean** expression (or an expression that can be easily converted to boolean). Each member of the current node-set is tested against the boolean expression and kept if the expression is true; otherwise, it is rejected. A predicate is enclosed in square brackets, `[]`. Have a look at the following location path:

```
list/item/currentPrice[@currency="EUR"]
```

During evaluation, all `currentPrice` elements in AuctionItemList.xml are in the selected node-set. Then, the `@currency="EUR"` predicate is evaluated and the `currentPrice` elements whose currencies do not contain the EUR value are rejected.

Predicates can also use the relational operators >, <, >=, <=, and `!=` . They can also use boolean operators, as you'll see in Expressions .

Lab: Location paths

Now that I have explained what location paths are, your task is to modify AuctionItemSummary-Base.xsl to produce the following output -- specifically, a table containing only the items with currency listed in U.S. dollars:

Figure 4. Table containing auction items in U.S. dollars
Note: You need to replace the value of the `select` attribute inside the `list` template with the correct location path. Use single quotation marks (`'`) inside a string enclosed in double quotation marks (`"`). I will talk more about this in Expressions .

A solution to this is AuctionItemSummary-Section5.xsl. Change the second line of AuctionItemList.xml to refer to AuctionItemSummary-Section5.xsl, and open AuctionItemList.xml in your Web browser.

Location paths are a subset of the more general XPath expressions. An expression refers not only to a set of nodes (location paths), but can also return a boolean, a number, or a string.

Section 6. Expressions

Booleans

A **boolean** expression can have one of two values: true or false.

XPath defines the `and` and `or` operators. With `and`, the left operand is evaluated first: If it is false, the expression returns false; otherwise, the right operand is evaluated and determines the result of the expression. With `or`, the left operand is evaluated and if true, the expression returns true; otherwise, the right operand is evaluated and determines the value of the expression. As an example, the boolean expression `type="BidOnly"` evaluates to true in the context of the second `item` element of AuctionItemList.xml.

XPath defines the following operators:

- = means "is equal to"

- `!=` means "is not equal to"
- < means "is less than"
- <= means "is less than or equal to"
- > means "is greater than"
- >= means "is greater than or equal to"

For example, the boolean expression `bidIncrement != 10` returns true in the context of the first `item` element in AuctionItemList.xml and false in the context of the second `item` element.

The = operator, when applied to nodes, tests whether two nodes have the same value, not whether they are the same node. This can be used to compare attribute values. For example, `item[@type = @private]` selects items whose `type` and `private` attributes have the same value.

When an XPath expression is contained in an XML document, the well-formedness rules of XML 1.0 need to be followed, and any < or <= characters must be quoted using `<` and `<=`, respectively. For example, the XPath expression `bidIncrement < 5` is valid in XPointer but needs to be written as `bidIncrement < 5` if it is to be contained in an XSLT document.

Conversions happen when operands of a boolean expression are not of the same type (node-set, numbers, strings). Refer to the XPath 1.0 specification for details.

Numbers

An XPath **number** is a double precision 64-bit floating-point number. XPath numbers include the "Not-a-Number" `NaN` value, positive and negative infinity, and positive and negative zero.

Numeric operators provided by XPath are: + (addition), `-` (subtraction), * (multiplication), `div` (division), and `mod` (remainder from truncating division).

Numeric operators convert operands to numbers if needed, as if they were using the `number` function (described in Function library).

Note: The subtraction (`-`) operator has to be preceded by whitespace because XML allows "`-`" characters in strings.

Here are a few examples of XPath numeric expressions:

- 7 + 3 returns 10

- 7 - 3 returns 4
- 7 * 3 returns 21
- 7 div 3 returns 2.3333333333333335
- 7 mod 3 returns 1

Note: An asterisk (`*`) can be interpreted as the wild card character or as the multiplier character. XPath defines lexical rules to eliminate ambiguities (see the XPath 1.0 specification for details). However, a new operator was introduced for the division character, `div`, because the forward slash (`/`) is used to separate location steps.

Strings

XPath **strings** are a sequence of valid XML 1.0 (Unicode) characters -- for example, `Miles Smiles album, CD`.

Strings in XPath are enclosed in quotation marks (`'` or `"`). When an XPath string is contained in an XML document and contains quotation marks, you have to use one of the two following options:

- Quote them using `&apos;` or `&quot;`, respectively. For example, `description = 'New 256m &quot;USB&quot; MP3 player'`.
- Use single quotation marks (`'`) if the expression is enclosed in double quotation marks (`"`), and vice-versa. For example, `'New 356m "USB" MP3 player'`.

XPath provides useful functions for dealing with strings, as described in Function library .

Lab: Expressions

Now that I have explained XPath expressions, your task is to modify AuctionItemSummary-Base.xsl to produce the following output -- a table containing all the items whose auction finishes within the hour:

Figure 5. Table containing auctions finishing in the hour
Note: `endOfAuction` is the time remaining until the end of the auction, in minutes. You need to modify the same `select` attribute as in Location paths .

A solution to this is AuctionItemSummary-Section6.xsl. Change the second line of AuctionItemList.xml to refer to AuctionItemSummary-Section6.xsl, and open

AuctionItemList.xml in your Web browser.

Section 7. Function library

Function Library

XPath defines a set of functions called the **core function library**. Each function is defined by three artifacts:

- A function name
- A return type (mandatory, no void)
- The type of the arguments (zero or more, mandatory, or optional)

You may find functions useful inside predicates and expressions. Other specifications, such as XSLT, extend this function set. Functions are divided into four groups, which I discuss in the rest of this section:

- Node-set functions
- String functions
- Boolean functions
- Number functions

Node-set functions

Node-set functions provide information on a set of nodes (one or more nodes). Useful node-set functions include:

- **`last()`** - Returns a number called the **context size**, which is the number of nodes in a given context. This is different from the last node.
- **`position()`** - Returns a number called the **context position**, which is the position of the current node in the set (list) of nodes in a given context. For example, you can test whether you are dealing with the last node of a set with the expression `position()=last()`.
- **`count(node-set)`** - Returns the number of nodes in the argument

node-set. For example, in the context of the AuctionItemList.xml document, `count(//item)` returns the number of `item` elements, which is 7.

- **`id(object)`** - Returns a node-set, the result of selecting elements by their unique id declared as type ID in a DTD. Because I don't use a DTD in AuctionItemList.xml, the result node-set is always empty for this example. `Id("ItemId0001")` returns an empty node-set.

XPath also defines three other functions related to node names and namespaces:

- `local-name()`
- `namespace-uri()`
- `name()`

Refer to section 4.1 of the XPath 1.0 specification for details.

String functions

With string functions, you can manipulate strings. Useful string functions include:

- **`string()`** - Converts the argument object or the context node to a string. Valid arguments are a node-set, a number, a boolean, or any other type -- but in the last case, the conversion result is less predictable. It is recommended to use XSLT's `format-number` function to convert numbers to strings, or XSLT's `xsl:number` element for presenting to users.
- **`concat()`** - Takes two or more strings as arguments and returns the concatenation of them. For example, `concat("Original ","recording ","Blue Train LP record")` returns `"Original recording Blue Train LP record"`.
- **`starts-with()`** - Returns true if the first argument string starts with the second argument string; false otherwise. For example, `starts-with("Miles Smiles album, CD", "Miles")` returns true.
- **`contains()`** - Returns true if the first argument string contains the second argument string; false otherwise. For example, `contains("Miles Smiles album, CD", "album")` returns true.

Other XPath string functions are **`substring()`** , **`substring-before()`** , **`substring-after()`** , **`string-length()`** , **`normalize-space()`** , and **`translate()`** . Refer to section 4.2 of the XPath 1.0 specification for details.

Boolean functions

Boolean functions are used to convert an object or a string to either true or false, or to get the true or false values directly. The boolean functions are:

- **`boolean()`** - Returns the conversion to boolean of the object passed as an argument, according to the following rules: A number is true if different from zero or `NaN`; a node-set or a string are true if not empty. Other types of objects are converted in a less predictable way.
- **`not()`** - Returns true if the boolean passed as argument is false; false otherwise.
- **`true()`** and **`false()`** - Return true or false, respectively. These functions are useful because true and false are seen as normal strings in XPath, and not the true and false values.
- **`lang()`** - Returns true if the language of the context node is the same or a sub-language of the string argument is specified; false otherwise. The language of the context node is defined by the value of the `xml:lang` attribute. For example, `lang("en")` returns false on any node of the tree for AuctionItemList.xml because the `xml:lang` attribute is not specified.

Number functions

Number functions are XPath's numeric functions, and they all return numbers. They are:

- **`number()`** - Converts the optional object argument (or the context node if no argument is specified) to a number, according to the following rules:
  - Boolean true is converted to 1, false to 0.
  - A string is converted to a meaningful number.
  - A node-set is first converted to a string and then the string converted to a number.

  Other types of objects are converted in a less predictable way. For example, `number("250")` returns 250 and `number("miles1965")` returns `NaN`.
- **`sum()`** - Returns the sum of all nodes in the node-set argument after the `number()` function has been applied to them.
- **`floor()`** - Returns the largest integer number that is not greater that the number argument. For example, `floor(2.75)` returns 2.

- **`ceiling()`** - Returns the smallest integer number that is not less than the number argument. For example, `ceiling(2.75)` returns 3.
- **`round()`** - Returns the integer number that is closest to the number argument. For example, `round(2.75)` returns 3.

Lab: Function library

Now that I have explained XPath functions, your task is to modify AuctionItemSummary-Base.xsl to produce the following output -- a table containing only the new auction items:

Figure 6. New auction items only
Note: Such items will contain the string `New` or `NEW` in their description. You need to modify the same `select` attribute as in Location paths and Expressions .

A solution to this is AuctionItemSummary-Section7.xsl. Change the second line of AuctionItemList.xml to refer to AuctionItemSummary-Section7.xsl and open AuctionItemList.xml in your Web browser.

Section 8. Tutorial wrap-up

XPath 2.0

XPath 2.0 is a superset of XPath 1.0 and currently a W3C Working Draft. Two W3C working groups are working on version 2.0 of XPath: the XML Query Working Group and the XSL Working Group. XPath 2.0 has more power and is more robust because it supports a broader set of data types. This is because XPath 2.0 values use XML Schema types rather than simple strings, numbers, or booleans. XPath 2.0 is backward-compatible so that 1.0 expressions behave the same in 2.0, with exceptions listed in the specification.

See Resources for more information on XPath 2.0.

Summary

In this tutorial, you learned that XPath is a language that's used to address parts of XML documents. You saw how XPath relates to other XML technologies, such as XSLT and XPointer. You saw what XPath expressions are, including the special

case of expressions called location paths. You also had an overview of the XPath function library and the new features of the upcoming XPath 2.0.

Downloads

Description	Name	Size	Download method
Article source code	XPathTutorial.zip	4KB	HTTP

Information about download methods

Resources

Learn

- Read the XML Path Language (XPath) version 1.0 Recommendation at the W3C site.
- Take a look at the XPath 2.0 specification (draft).
- Find out about new features in the upcoming XPath 2.0 in the *developerWorks* article "XML for Data: What's new in XPath 2.0?" (September 2002).
- Review your XML basics with Doug Tidwell's tutorial "Introduction to XML" (*developerWorks*, August 2002).
- Learn more about transforming XML documents -- read the XSL Transformations (XSLT) 1.0 specification.
- Check out the W3C XML Schema Recommendation.
- Try the XML Linking Language (XLink) to insert elements into XML documents in order to create and describe links between resources.
- Explore the XML Pointer Language (XPointer), an extension to XPath that provides addressing into XML documents and their internals.
- Find more XML resources on the *developerWorks* XML zone.
- Find out how you can become an IBM Certified Developer in XML and related technologies.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Build your next development project with IBM trial software, available for download directly from developerWorks.

About the author

Bertrand Portier

Bertrand Portier is a software engineer working in the IBM Software Group Services organization. He works with IBM® WebSphere® customers to provide solutions to their business problems. Before that, he used to work on product development in the Web services area. You can contact Bertrand at portier@uk.ibm.com.