



# XPath for .NET Developers

**Author:** [Darshan Singh](#) (Managing Editor, PerfectXML.com)

**Last Updated:** May 12, 2003

**[Code Download.](#)**

---

## Abstract

XML Path Language (or XPath) is a W3C standard that primarily allows identifying parts of an XML document. This means that if you have an XML document and would like to locate one or more nodes, you can use XPath to do that. In addition, XPath is also used for numerical calculations, string manipulations, testing boolean conditions, etc.

XPath is used by various other W3C specifications such as XSLT, XQuery, XPointer, and XML Schema. And hence understanding of XPath is one of the basic skills that every XML developer should acquire.

**XPath 1.0** became W3C recommendation on 16th November, 1999 (<http://www.w3.org/TR/xpath>). W3C continued working on further enhancements to the XPath specification, and as of today (May 2003) the **XPath 2.0** is in the W3C Working Draft status (<http://www.w3.org/TR/xpath20>).

In this article, you'll learn about XPath and how to use it in .NET applications. The Microsoft .NET Framework supports XPath 1.0 W3C recommendation. The classes in System.Xml and System.Xml.XPath namespaces allow executing XPath queries and working with the result sets. Before we look at these classes and examples, let's first review XPath basic concepts and terminology.

This article assumes that you are familiar with basic XML concepts, including XML Namespaces, and also have some familiarity with .NET Framework programming using C# and Visual Basic .NET.

## XPath Primer

The XPath specification broadly covers following four topics:

1. **Data Model:** This section describes the general concepts and terms used in XPath.
2. **Location Paths:** This section details the constructs and syntax used for addressing parts of XML document. A location path is used to address a certain `node-set` of a document. The location path syntax is similar to other hierarchical notations used in computer applications such as URIs, file/folder paths, etc; and location path can be either absolute or relative. Location path example: `/Customer/Orders[position()=2]`

3. **Expressions:** This section describes the most basic XPath construct: an expression. Location paths (explained above) are special case of XPath expressions. Expressions are made up of operands and operators. Expression is evaluated to yield a node-set (an unordered collection of nodes without duplicates), a boolean value (true or false), a floating point number, or a string. Expression example: `20 mod 3`
4. **Functions:** This section discusses about 27 functions that are divided into four categories: Node Set Functions, String Functions, Boolean Functions, and Number Functions. Each function takes zero or more arguments and returns a single result. Function example: `concat(FirstName, ' ', LastName)`

## Data Model

XPath considers XML document as a logical **tree of nodes**. It defines seven types of nodes:

1. **Root Node:**

This node is the root of the tree. Each XML document has exactly one root node and it contains document element, processing instructions, and comment nodes. It is important to note the difference between the root node and the document element. Root node is not really visible in the XML document, it is the topmost parent. The topmost (visible) element in an XML document (the document element) is the child of root node. In addition the root node does not have an expanded-name (discussed later).
2. **Element Node:**

Each element in an XML document is represented by an element node. Element nodes may contain other element nodes, comment nodes, processing instruction nodes, and text nodes (element's content). Each element node has an expanded-name, which is evaluated in accordance with the XML Namespaces recommendation.
3. **Attribute Node:**

For each attribute of an element, there is an attribute node. Although the element node is the parent of all of its attribute nodes, the converse is not true - that is, attribute nodes are not treated as children of the element node. Each attribute node has an expanded-name, which is evaluated in accordance with the XML Namespaces recommendation.
4. **Text Node:**

Character data inside element (including each character within a CDATA section, if any) is termed as text node. A text node always has at least one character of data. A text node does not have an expanded-name.
5. **Namespace Node:**

Each element has an associated set of namespace nodes, one for each distinct namespace prefix that is in scope for the element (including the xml prefix, which is implicitly declared by the XML Namespaces Recommendation) and one for the default namespace if one is in scope for the element. Like with the attribute nodes, the element is the parent of each of these namespace nodes; however, a namespace node is not a child of its parent element. A namespace node has an expanded-name.
6. **Processing Instruction Node:**

There is a processing instruction node for every processing instruction in the XML document (except the PI in the document type declaration). Also note that since the

XML declaration (<?xml version=...?>) is not a processing instruction. Therefore, there is no processing instruction node corresponding to the XML declaration. A processing instruction node has an expanded-name.

7. **Comment Node:**

For every comment in an XML document, there is a comment node (except for any comment that occurs within the document type declaration). A comment node does not have an expanded-name.

## Location Paths

A location path essentially selects a set of nodes relative to the context node. An absolute location path begins with a slash character.

Location path examples:

**/invoice/billto/address/line1**

Selects line1 element(s).

**//ImportDetails**

Selects ImportDetails element(s).

**News/@Date**

Selects Date attribute.

**/contacts/contact[not(@type = preceding-sibling::contact/@type)]/@type**

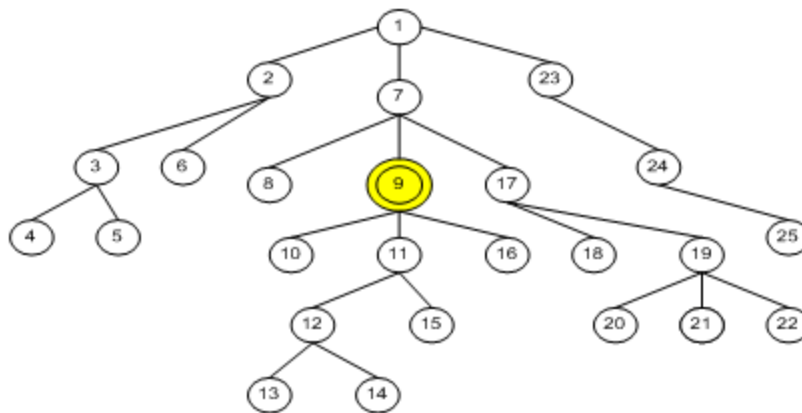
This absolute location path finds all type attribute nodes of contact parent node where none of the previous contact nodes has the type attribute with the same value as current type attribute. In essence, it is used to find the DISTINCT type attribute values.

Location path is generally divided into several steps, which are separated by a forward slash (/). Each step is known as a Location Step. The core building blocks of a location step includes **axes** (for example: preceding-sibling), **node tests** (for example: text()), and **predicates** (for example: contact[not(@type = preceding-sibling::contact/@type)]):

1. **Axis:**

It defines which nodes are selected starting from the context node. In other words, an axis specifies the tree relationship between the nodes selected by the location step and the context node.

Consider the following sample XML data model tree (all elements, yellow node 9 is the context node):



Assuming that node 9 above is the context node, the following table lists various axis names and respective result nodes in above XML tree:

Axis Name	Selected Elements
<b>ancestor</b>	1 and 7
<b>ancestor-or-self</b>	1, 7, and 9
<b>child</b>	10, 11, and 16
<b>descendant</b>	10-through-16
<b>descendant-or-self</b>	9-through-16
<b>following</b>	17-through-25
<b>following-sibling</b>	17
<b>parent</b>	7
<b>preceding</b>	2-through-6, and 8
<b>preceding-sibling</b>	8
<b>self</b>	9

There are two more XPath axes: `attribute` and `namespace`, used to select attribute and namespace nodes, respectively.

**Click here** to download a sample VB 6.0 application that uses MSXML 4.0 and illustrates the XPath Axes concept.

## 2. Node Tests:

Every axis in the location step selects a node-set, and then node test is applied to all these nodes, which potentially reduces the number of nodes in the resultant node-set. For instance, if the axis `child::` selects child nodes of the context node, then `child::text()` will select all text child nodes of the context node. `child::*` will select all element children of the context node, `attribute::*` will select all attributes of the context node. `child::contact` will select all child elements named 'contact' under the current context node, `child::comment()` will select all the comment children nodes of the current context node, and so on. In summary, node tests allow us to locate the node(s) by name or by type.

## 3. Predicates:

Predicates can be used to specify the filtering criteria. Predicate is applied to each node in the current node-set. If the node matches the predicate, that is if the predicate evaluates to true, the node remains in the resulting node-set, else it is removed from the node set. For instance the predicate `[position() = 1]` selects the first node in the node-set. If you understand the XPath expressions and functions, you can easily write very complex predicates to apply the appropriate filtering criteria.

XPath supports abbreviated syntax for various location paths. For instance, to select all attribute nodes, instead of writing `attribute::*`, you can write `@*`; instead of writing `self::node()`, you can write `.` (a single period); instead of writing `parent::node()`, you can write `..` (two period characters); instead of writing `descendant-or-self::node()`, you can write `//` (two forward slash characters); instead of writing `[position() = 5]`, you can write just `[5]`; also if you can completely omit the `child::` and still selects the exact or all child elements of the current context node.

# Expressions

An expression is the most basic construct of an XPath. Location paths, discussed above, are just one special case of XPath expressions. An XPath expression consists of operators and operands. XPath operators include negation (-), multiplication (\*), floating-point division (div), remainder (mod), addition (+), subtraction (-), less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), equal (=), not equal (!=), logical and (and), logical or (or), and union (|). As mentioned earlier, an XPath expression is evaluated to result in one of four types of objects: a node-set, a boolean value, a floating-point number, or a string.

# Functions

XPath 1.0 defines 27 functions categorized under following four categories:

## 1. String Functions

Function	Parameters	Return Type	Description
concat	string, string, string*	string	Concatenates two or more strings
contains	string, string	boolean	Tests for containment of one string in another
normalize-space	string?	string	Normalizes whitespace in a string
starts-with	string, string	boolean	Tests if one string starts with another
string	object?	string	Converts to string
string-length	string?	number	Returns number of characters in a string
substring	string, number, number?	string	Extracts a substring from a string
substring-after	string, string	string	Selects after a matching string
substring-before	string, string	string	selects before a matching string
translate	string, string, string	string	Replaces characters in a string

## 2. Number Functions

Function	Parameters	Return Type	Description
ceiling	number	number	Rounds up a number
floor	number	number	Rounds down a number
number	object?	number	Converts to a number
round	number	number	Rounds to the next closest integer number
sum	node-set	number	Sums the values of all nodes

## 3. Boolean Functions

Function	Parameters	Return Type	Description
boolean	object	boolean	Converts to a boolean value
false		boolean	Returns false
lang	string	boolean	Tests for language of nodes
not	boolean	boolean	Inverts a boolean value
true		boolean	Returns true

#### 4. Node Set Functions

Function	Parameters	Return Type	Description
count	node-set	number	Returns the number of nodes
id	object	node-set	Returns node-set with element selected by ID
last		number	Returns a numeric pointer to the last set member
local-name	node-set?	string	Returns the local part of the first node
name	node-set?	string	Returns the expanded name of the first node
namespace-uri	node-set?	string	Returns the namespace URI of the first node
position		number	Returns the numeric pointer to the context position

With this introduction to XPath, let's now see how to evaluate XPath expressions in .NET.

## XPath and .NET

If you have used MSXML DOM, you might be familiar with `selectNodes` and `selectSingleNode` methods. These methods, given the XPath expression, are used to select a node-set from the loaded XML document. As mentioned earlier, node-set is just one of four types of objects that can be returned when an XPath expression is evaluated (the other three include: string, boolean, and a number). MSXML does not provide direct way of evaluating XPath expression that returns string, boolean, or number. In other words, MSXML `selectNodes` and `selectSingleNode` methods can only be used to get the node-set.

The good news is that this is fixed in .NET, and XPath evaluation in .NET can result all the four types of objects: string, number, boolean, and node-set.

To understand the XPath support in .NET, carefully read the following scenarios:

- If you already have a XML document loaded using DOM, or if you need to edit/write XML document, in addition to querying, you can use any class derived from **XmlNode** (Such as `XmlDocument` Or `XmlDataDocument`) and use **SelectNodes** and **SelectSingleNode** methods. These methods can only return the object of type node-set, and can not be used if the XPath expression is returning number, boolean, or a string value. The `SelectNodes` method returns `XmlNodeList`, whereas `SelectSingleNode` returns `XmlNode`.

Now, if the XPath expression is returning object type other than node-set, you can first call **CreateNavigator** on any `XmlNode`-derived class and get an instance of **XPathNavigator** class (defined in `System.Xml.XPath` namespace), and use this object to evaluate the XPath expression to get a typed result. In addition to getting string, number, or boolean result value, the `XPathNavigator` class can also be used for other things, such as to do sorting, to execute pre-compiled XPath expression (little bit optimization), and so on.

- If your goal is to only query the XML document, and you'll not be updating the XML document, you can use **XPathDocument** class defined in `System.Xml.XPath` namespace. This class is optimized for XPath data model, and it facilitates fast, read-only, cache-optimized XPath processing. This class (like `XmlNode`-derived classes) supports `CreateNavigator` function that can be used to get the `XmlPathNavigator` instance. Both, `XmlNode` and `XPathDocument` implement an interface named **IXPathNavigable**, which has just one method named `CreateNavigator`.

The `XPathNavigator` methods, `Select`, `SelectChildren`, `SelectAncestors`, `SelectDescendants`, and `Evaluate` return an instance of a class `XPathNodeIterator`. This class (`XPathNodeIterator`) provides methods that support iterating over a set of selected nodes. As mentioned in the previous statement, the `XPathNavigator` method `Evaluate` can return `XPathNodeIterator` and hence work with XPath expressions that return node-sets. In addition, `Evaluate` method can also be used if the XPath expression evaluates to string, number, or a boolean value.

The `Compile` method in `XPathNavigator` class can be used to pre-compile an XPath expression. This method returns an instance of type `XPathExpression`, which can also be used to determine the XPath expression result type (using `ReturnType` property).

- Use `XmlNamespaceManager` class when working with XPath and namespaces.
- Use `XPathException` to handle exception raised during processing XPath expression.

## Example 1: XmlDocument and SelectNodes method

Let's start with a very simple example. This sample C# console application accepts two command line parameters: XML file to load and XPath expression string; it loads the specified XML document using `XmlDocument` class, selects the nodes based on the provided XPath expression, and prints the resultant node-set values. This sample application does not support XML namespaces.

```
using System;
using System.Xml;

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        if(args.Length < 2)
        {
            Console.WriteLine("Incorrect number of parameters.");
            goto endApp;
        }

        try
        {
            //Load the XML document
            XmlDocument doc = new XmlDocument();
            doc.Load(args[0]);

            //Select the nodes
            XmlNodeList resultNodes = doc.SelectNodes(args[1]);
            Console.WriteLine("{0} matching nodes found...",
resultNodes.Count);

            //Print the node values
            int iCount=0;
            foreach(XmlNode aNode in resultNodes)
```

```

        Console.WriteLine("Node {0} value: {1}",
            ++iCount, aNode.InnerText);
    }
    catch(Exception exp)
    {
        Console.WriteLine("Error: " + exp.ToString());
    }
}

endApp:
    Console.Write("Press Enter to continue...");
    Console.ReadLine();
}
}

```

The above code first checks if less than two arguments are passed, in which case, it prints the error message and the application exits. The code first loads the XML document (the first command line parameter, XML file URL) using `XmlDocument load` method. It then evaluates the XPath expression (the second command line parameter) using `SelectNodes` method, and prints the count of result nodes, followed by text value of each result node.

Let's say you created the above console application class file as part of project named `XPathDemo1`, save and build, run the application on command line as follows:

```
XPathDemo1.exe c:\NWCustomers.xml /customers/row/@CompanyName
```

**Download** the code for this article, and save the included `NWCustomers.xml` on your `C:\` root. The above command line execution should result in the following output:

```

6 matching nodes found...
Node 1 value: Alfreds Futterkiste
Node 2 value: Ana Trujillo Emparedados y helados
Node 3 value: Antonio Moreno Taquería
Node 4 value: Around the Horn
Node 5 value: Berglunds snabbköp
Node 6 value: Blauer See Delikatessen
Press Enter to continue...

```

Also try:

```
XPathDemo1.exe http://www.PerfectXML.com/books.xml /catalog/book/author
```

You should see the output as:

```

12 matching nodes found...
Node 1 value: Gambardella, Matthew
Node 2 value: Ralls, Kim
Node 3 value: Corets, Eva
Node 4 value: Corets, Eva
Node 5 value: Corets, Eva
Node 6 value: Randall, Cynthia
...

```

Let's say, you wanted to find out if there are any books with price greater than 40\$. You can use the above sample application and pass the XPath expression `"/catalog/book/price[. >`

40]" and it would return the price nodes where the node value is higher than 40.

Let's update the above sample, use **XPathNavigator** and **XPathExpression** classes from the **System.Xml.XPath** namespace and build yet another console application that can evaluate XPath expressions and return any of four types of objects: string, boolean, number, and a node-set. Like previous sample, this console application also takes two parameters: XML file URL, and the XPath expression.

```
using System;
using System.Xml;
using System.Xml.XPath;

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        if(args.Length < 2)
        {
            Console.WriteLine("Incorrect number of parameters.");
            goto endApp;
        }

        try
        {
            //Load the XML document
            XmlDocument doc = new XmlDocument();
            doc.Load(args[0]);

            //Create XPathNavigator
            XPathNavigator xpathNav = doc.CreateNavigator();

            //Compile the XPath expression
            XPathExpression xpathExpr = xpathNav.Compile(args[1]);

            //Display the results depending on type of result
            switch(xpathExpr.ReturnType)
            {
                case XPathResultType.Boolean:
                    Console.WriteLine("Boolean value: {0}",
                        xpathNav.Evaluate(xpathExpr));
                    break;

                case XPathResultType.String:
                    Console.WriteLine("String value: {0}",
                        xpathNav.Evaluate(xpathExpr));
                    break;

                case XPathResultType.Number:
                    Console.WriteLine("Number value: {0}",
                        xpathNav.Evaluate(xpathExpr));
                    break;

                case XPathResultType.NodeSet:
```

```

        XPathNodeIterator nodeIter =
xpathNav.Select(xpathExpr);

        Console.WriteLine("Node-set count: {0}",
            nodeIter.Count);

        while(nodeIter.MoveNext())
            Console.WriteLine(nodeIter.Current.Value);
        break;

        case XPathResultType.Error:
            Console.WriteLine("XPath expression {0} is invalid.",
                args[1]);
            break;
    }
}
catch(Exception exp)
{
    Console.WriteLine("Error: " + exp.ToString());
}

endApp:
    Console.Write("Press Enter to continue...");
    Console.ReadLine();
}
}
}

```

Things to note in the above code:

- It takes two command line parameters, the XML file URL and the XPath expression.
- After the document is loaded using DOM (`XmlDocument.load`), it calls `CreateNavigator` method on it, which returns `XPathNavigator`. This instance is then used to compile the input XPath expression, find out what type of result (string, number, etc.) would be when the expression is evaluated, and then finally to **evaluate** the expression or **select** the nodes (depending on the result type).
- The `Evaluate` function is called when the result type is number, string, or boolean, and `Select` method is called when the result type is node-set.
- For node-set return value, the `Select` method returns `XPathNodeIterator`, which we loop over to get the individual node values. `XPathNodeIterator` basically provides an iterator over the selected nodes.

Save the above code as part of console application named `XPathDemo2.exe`, give the first parameter as <http://www.PerfectXML.com/books.xml>, and try out following XPath expressions as second parameter value (enclosed in double quotes if the XPath expression contains a space)

XPath Expression	Result	Description
"count(/catalog/book/price[. > 40]) > 0"	Boolean value: True	Checks if <i>count</i> of books having price

		greater than 40 or not.
"concat(substring(/catalog/book[2]/author, 1, 4), '...')"	String value: Rall...	Gets first four characters for second book's author node, and then concatenates with '...'
"count(/catalog/book/price[. > 40])"	Number value: 2	Returns count of books having price greater than 40
"/catalog/book[@id='bk103']/*"	Node-set count: 6 Corets, Eva Maeve Ascendant Fantasy 5.95 2000-11-17 After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.	Returns all children <i>element</i> nodes for book having id equals bk103.
"/catalog/book/author"	Node-set count: 12 Gambardella, Matthew Ralls, Kim Corets, Eva Corets, Eva Corets, Eva Randall,	Return all author nodes

	Cynthia Thurman, Paula Knorr, Stefan Kress, Peter O'Brien, Tim O'Brien, Tim Galos, Mike	
"/catalog/book[not(author = preceding-sibling:book/author)]/author"	Node-set count: 9 Gambardella, Matthew Ralls, Kim Corets, Eva Randall, Cynthia Thurman, Paula Knorr, Stefan Kress, Peter O'Brien, Tim Galos, Mike	Return DISTINCT author nodes

We used `XmlDocument` in the above examples; as an exercise write some sample application that loads the XML document using `XPathDocument`, call `CreateNavigator` on it and execute XPath expressions. Also try loading `XmlDataDocument` by running an ADO.NET query, and then executing XPath expressions on this `XmlDataDocument` instance.

## Summary

The importance and popularity of XPath can be gauged by the fact that many other XML specifications (such as XSLT and XQuery) are heavily relying on it, that it is getting more and more tools support, and that W3C is fast working towards 2.0 of this specification. Understanding XPath is one of the essential skills that every XML developer should have. And once you know what XPath is and how it works, using it in various APIs, such as in MSXML or .NET is just matter of understanding the API/classes created to support XPath.

This article introduced basic XPath concepts, followed by how XPath is supported in .NET via `System.Xml` and `System.Xml.XPath` namespace classes, and finally couple of examples were presented as proof of concept.

## Further Reading

- [Things to Know and Avoid When Querying XML Documents with XPath](#)
- [Roadmap for Executing XPath Queries in .NET Applications](#)
- [Adding Custom Functions to XPath](#)
- [XPath Querying Over Objects with ObjectXPathNavigator](#)
- [XPath, XSLT, and other XML Specifications](#)
- [Addressing Infosets with XPath](#)
- [Introducing XPath 2.0](#)

- **Writing XML Providers for Microsoft .NET**
- **XPath Selections and Custom Functions, and More**
- **MSXML and XPath**
- **XPath & XSLT Samples by Aaron Skonnard**