



TopXML reference – May 2002

Table of Contents

The Microsoft DOM	1
<u>DOM objects</u>	1
<u>DOMDocument</u>	2
<u>XMLDOMElement</u>	3
<u>XMLDOMNode</u>	3
<u>XMLDOMNodeList</u>	5
<u>XMLDOMNamedNodeMap</u>	5
<u>XMLDOMCDATASection</u>	6
<u>XMLDOMAttribute</u>	6
<u>XMLDOMDocumentType</u>	7
<u>XMLDOMEntity</u>	8
<u>XMLDOMProcessingInstruction</u>	8
<u>XMLDOMParseError</u>	9
<u>XMLHttpRequest</u>	10
<u>DOM object properties</u>	10
<u>async</u>	15
<u>Attributes</u>	16
<u>childNodes</u>	18
<u>data</u>	22
<u>docType</u>	24
<u>documentElement</u>	25
<u>length</u>	28
<u>namespaceURI</u>	29
<u>nodeName</u>	30
<u>nodeType</u>	34
<u>The 12 Node types</u>	36
<u>nodeValue</u>	45
<u>nodeTypeString</u>	47
<u>nodeValue</u>	48
<u>ondataavailable</u>	51
<u>ownerDocument</u>	52
<u>parentNode</u>	52
<u>parsed</u>	53
<u>parseError</u>	55
<u>prefix</u>	56
<u>preserveWhiteSpace</u>	57
<u>resolveExternals</u>	58
<u>tagName</u>	59
<u>text</u>	59
<u>url</u>	61
<u>validateOnParse</u>	62
<u>value</u>	62
<u>DOM object methods</u>	63

Table of Contents

The Microsoft DOM

abort()	65
appendChild()	66
cloneNode()	70
createAttribute(), createCDATASection(), createComment(), createElement(), createEntityReference(), createProcessingInstruction(), createTextNode()	71
createNode()	76
getAttribute()	79
getAttributeNode()	80
getElementsByTagName()	80
getNamedItem()	82
hasChildNodes()	83
insertBefore()	85
load()	86
loadXML()	86
nextNode()	87
nodeFromID()	89
removeAttribute()	90
removeAttributeNode()	91
removeChild()	92
removeNamedItem()	94
replaceChild()	95
reset()	96
save()	97
selectNodes()	99
selectSingleNode()	100
send()	102
setAttribute()	102
setAttributeNode()	103
setNamedItem()	104
transformNode()	105
transformNodeToObject()	110

XPath Reference.....111

General intro	111
Axes	111
ancestor	111
ancestor-or-self	111
attribute	111
child	112
descendant	112
descendant-or-self	112
following	112

Table of Contents

XPath Reference

<u>following-sibling</u>	113
<u>namespace</u>	113
<u>parent</u>	113
<u>preceding</u>	113
<u>preceding-sibling</u>	113
<u>self</u>	114
<u>Node tests</u>	114
<u>*</u>	114
<u>comment()</u>	114
<u>literal name</u>	114
<u>node()</u>	114
<u>processing-instruction(name?)</u>	114
<u>text()</u>	114
<u>Functions</u>	115
<u>boolean</u>	115
<u>ceiling</u>	115
<u>concat</u>	115
<u>contains</u>	115
<u>count</u>	116
<u>false</u>	116
<u>floor</u>	116
<u>id</u>	116
<u>lang</u>	117
<u>last</u>	117
<u>local-name</u>	117
<u>name</u>	117
<u>namespace-uri</u>	118
<u>normalize-space</u>	118
<u>not</u>	118
<u>number</u>	118
<u>position</u>	119
<u>round</u>	119
<u>starts-with</u>	119
<u>string</u>	119
<u>string-length</u>	119
<u>substring</u>	120
<u>substring-after</u>	120
<u>substring-before</u>	120
<u>sum</u>	121
<u>translate</u>	121
<u>true</u>	121
<u>A few examples of XPath expressions</u>	121

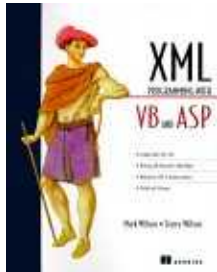
Table of Contents

XSLT Reference	123
<u>Elements</u>	123
<u>xsl:apply-imports</u>	123
<u>xsl:apply-templates</u>	123
<u>xsl:attribute</u>	124
<u>xsl:attribute-set</u>	124
<u>xsl:call-template</u>	125
<u>xsl:choose</u>	125
<u>xsl:comment</u>	125
<u>xsl:copy</u>	126
<u>xsl:copy-of</u>	126
<u>xsl:decimal-format</u>	127
<u>xsl:element</u>	128
<u>xsl:fallback</u>	128
<u>xsl:for-each</u>	129
<u>xsl:if</u>	129
<u>xsl:import</u>	130
<u>xsl:include</u>	130
<u>xsl:key</u>	131
<u>xsl:message</u>	131
<u>xsl:namespace-alias</u>	132
<u>xsl:number</u>	132
<u>xsl:otherwise</u>	134
<u>xsl:output</u>	134
<u>xsl:param</u>	135
<u>xsl:preserve-space</u>	136
<u>xsl:processing-instruction</u>	136
<u>xsl:sort</u>	137
<u>xsl:strip-space</u>	137
<u>xsl:stylesheet</u>	138
<u>xsl:template</u>	139
<u>xsl:text</u>	140
<u>xsl:transform</u>	140
<u>xsl:value-of</u>	141
<u>xsl:variable</u>	141
<u>xsl:when</u>	142
<u>xsl:with-param</u>	142
<u>Functions</u>	143
<u>current</u>	143
<u>document</u>	143
<u>element-available</u>	143
<u>format-number</u>	143
<u>function-available</u>	144

Table of Contents

<u>XSLT Reference</u>	
<u>generate-id</u>	144
<u>key</u>	144
<u>system-property</u>	145
<u>unparsed-entity-url</u>	145
<u>Inherited XPath Functions</u>	145
<u>Types</u>	145
<u>Worked code samples</u>	146
<u>Example: Combining and intersecting two nodesets</u>	146
<u>Example: Creating a summary of author sales for a publisher</u>	148
<u>Example: Creating an HTML document with 'previous' and 'next' links</u>	150
<u>Example: Creating listboxes and checkboxes using parameters</u>	154
<u>Example: Creating listboxes and checkboxes using variables</u>	157
<u>Example: Generating a new stylesheet</u>	160
<u>Example: Numbering paragraphs and chapters</u>	164
<u>Example: Using different axes</u>	166
<u>Example: Whitespace preserving and stripping</u>	172
<u>Compatibility overview</u>	174
<u>Implementations covered</u>	177

The Microsoft DOM



This manuscript is an abridged version of a chapter from the [Manning Publications](#) book

[XML Programming with VB and ASP](#). This chapter looks at the Microsoft DOM object in quite some detail.

XML Programming with VB and ASP gives a VB/ASP developer a from-the-ground-up start to XML and XML programming in VB and ASP. Fun to read and easy going, this is a no-frills book which gets right to the point. More advanced topics include BizTalk, schemas, webclasses and XSL.

What this chapter of the book covers:

- Ø Descriptions of the Microsoft XML objects
- Ø Sample code for most of the methods and properties
- Ø Easy-reference table guide to the objects

DOM objects

First we need to explain the various, common DOM objects that you will use with the MSXML DOM Object.

We will focus on the examples of these XML DOM objects shown in table .

XML DOM Objects

Object name	Description
DOMDocument	This object represents the root of the XML file.
XMLDOMElement	This object represents each element in the DOM-Document, namely the root, root element, and each other element.
XMLDOMNode	This object represents a single Node in the document tree and includes support for data types, namespaces, DTDs, and XML Schemas.
XMLDOMNodeList	

	Use this object to access (by name) and iterate through the XMLDOMNode collection.
XMLDOMNamedNodeMap	Use this object to access and iterate through the attributes in an element.
XMLDOMCDATASection	This object represents a section in the value of an element that is closed in the CDATA section brackets, which are characters that cannot be parsed by the XML.
XMLDOMAttribute	This object represents a single attribute Node for a given element.
XMLDOMDocumentType	This object represents the Document Type (DTD) in an XML file.
XMLDOMEntity	This object represents an entity in the DTD section of the XML file.
XMLDOMProcessingInstruction	This object represents a processing instruction found in the XML file.
XMLDOMParseError	This object returns detailed information about the last error, including the line number, character position, and a text description.
XMLHttpRequest	This object enables you to establish a connection to a web server from your code and send put, get, and other standard HTML requests.

In our examples, we will refer to these objects using the names listed in table .

XML object naming conventions (continued)

XML object interface	Our naming convention
DOMDocument	objDOMDocument
XMLDOMNode	objXMLDOMNode
XMLDOMNodeList	objXMLDOMNodeList
XMLDOMNodeListMap	objXMLDOMNodeListMap
XMLDOMParseError	objXMLDOMParseError
XMLDOMElement	objXMLDOMElement
XMLDOMAttribute	objXMLDOMAttribute

DOMDocument

The DOMDocument object represents the root of the XML file. As the file is loaded into the DOMDocument, the XML file and its external references, such as DTDs, get validated by the DOMDocument.

The DOMDocument is the first *port of call* to the XML file. This is the only object that can be created. All the other objects, like the elements, can only be created or accessed from the DOMDocument object.

Example

This example loads an XML file into a DOMDocument object:

```
Dim objDOMDocument As DOMDocument
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

XMLDOMEElement

This object represents each element in the XML tree. The XMLDOMEElement includes support for manipulating the element and the attributes associated with the element.

The attributes associated with an element are added and manipulated via the XMLDOMEElement object.

Example

The following example returns the root element of the XML file from a DOMDocument:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMEElement As IXMLDOMEElement
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMEElement = objDOMDocument.documentElement
```

XMLDOMNode

This is one of the base objects of the DOM. Most of the other DOM objects inherit this object; therefore, you will see a lot of properties and methods repeated in each of these objects.

These objects are:

Ø DOMDocument

Ø XMLDOMAttribute

- Ø XMLDOMCDATASection
- Ø XMLDOMComment
- Ø XMLDOMDocumentFragment
- Ø XMLDOMDocumentType
- Ø XMLDOMEElement
- Ø XMLDOMEntity
- Ø XMLDOMEntityReference
- Ø XMLDOMNotation
- Ø XMLDOMProcessingInstruction
- Ø XMLDOMText
- Ø XTLRuntime

Although all these objects inherit methods and properties from the XMLDOMNode object, they will all also have properties and methods that are unique to their function. For example, the XMLDOMEElement object has extra methods for obtaining attribute information. The XMLDOMNode interface provides just the basic information, like the name of the Node, its text, etc.

To know which type of Node is currently being accessed, the `nodeType` property returns which type of Node you are referencing when using the XMLDOMNode object, which is explained in detail later in this chapter under the `nodeType` property.

Also, see the `docType` property for more information on *Dual interfaces*. This explains how to cast from one type of object to the other.

Example

The following example is the same as the example from the XMLDOMEElement, except it now uses the XMLDOMNode object to return the root element of the XML file from a DOMDocument.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMEElement As IXMLDOMNode
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMNode = objDOMDocument.documentElement
```

XMLDOMNodeList

The XMLDOMNodeList object is a collection of Nodes. Its methods allow us to iterate through all the children Nodes of a Node. You can use the For Each ... Next loop for this iteration. However, you can also choose to iterate through these Nodes using its method of nextNode().

This object is returned in the property, such as childNodes, or methods such as getElementByTagName() and selectNodes().

Example

The following example uses the childNodes property of the DOMDocument to return the children Nodes of the root element of the DOMDocument. (For more details see the childNodes property, which also shows an example of how to iterate through the XMLDOMNodeList collection.)

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNodeList As IXMLDOMNodeList
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMNodeList = objDOMDocument.documentElement.childNodes
```

XMLDOMNamedNodeMap

This is the other collection object in the DOM. It is used to iterate through the attributes for a specific element. It also allows you to manipulate the attribute collection for an element. To name a few, methods include getNamedItem(), removeNamedItem(), etc. The XMLDOMNamedNodeMap also supports namespaces as well.

Example

The following example returns the number of attributes found in the root element of a DOM object.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNamedNodeMap As IXMLDOMNamedNodeMap
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMNamedNodeMap = objDOMDocument.documentElement.Attributes
```

```
MsgBox "The number of attribute are " & objXMLDOMNamedNodeMap.length
```

XMLDOMCDATASection

The XMLDOMCDATASection object represents the content that is inserted in your XML between the CDATA section brackets, ![...]].

An XML example is:

```
<ADDRESS><![CDATA[#911 Somewhere Circle, Canberra, Australia]]></ADDRESS>
```

The XMLDOMCDATASection object inherits methods and properties from the XMLDOMText object; therefore, you will find the XMLDOMCDATASection object as a child Node of an XMLDOMText object. The parent XMLDOMText object will parse the contents of the CDATA section and return the full element text without the brack-ets. (See the nodeType property later in this chapter for more details on and exam-ples of the CDATA section.)

XMLDOMAttribute

The XMLDOMAttribute object represents an attribute for an element. Attributes are properties of an element, thus not really children of an element. As mentioned in the XMLDOMNamedNodeMap object, the collection of attributes for an element are returned in the XMLDOMNamedNodeMap object, which returns an XMLDOMAttribute object (or an XMLDOMNode object, as it inherits the XMLDOMNode interface).

The attributes are usually defined in the DTD or Schema of the XML file.

Example

In the following example we get an attribute object from the first child of the root element object, using the attributes name (which was defined in the DTD) to return the attribute object.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMEElement As IXMLDOMEElement
```

```
Dim objXMLDOMAttribute As IXMLDOMAttribute
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMElement = objDOMDocument.documentElement.firstChild
```

```
Set objXMLDOMAttribute = objXMLDOMElement.getAttribute–  
Node("PERSONID")
```

XMLDOMDocumentType

The XMLDOMDocumentType object contains information on all the entities and notations in a declared DTD of the XML file. This object is only found as a property, called the docType, of the DOMDocument object. This object and its properties are read-only. Therefore, you cannot add a DTD to a DOMDocument; it has to already have been declared in the XML file. You cannot add anything to the declared DTD for the XML file either.

Example

On our website, vbxml.com, there were quite a few discussions on how to add a DTD to a DOMDocument. With the current version of the msxml.dll, there was a bug if you added a DTD in a string and then used the loadXML() method.

The following example returns an XMLDOMDocumentType from the DOM-Document. We have added another line of code, objDOMDocument.resolveExternals = True, which is needed in order to instruct the DOMDocument to notice any external files associated with this XML file.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMDocumentType As IXMLDOMDocumentType
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.resolveExternals = True
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
Set objXMLDOMDocumentType = objDOMDocument.doctype
```

XMLDOMEntity

In your DTD file, you declare entities for *constants* that you want to use in your XML file.

For more information on entities, see chapter 2, “XML boot camp.”

In XML, five built-in entities (known as predefined entities) exist that all parsers can decipher (see table). Many parsers can automatically read more entities than these five, but the DOMDocument object currently only recognizes these five. The rest need to be defined in your DTDs.

Built-in entities that the DOMDocument automatically parses

Entity reference	Character that is represented
&	& ampersand
'	' apostrophe
>	> greater than
<	< less than
"	“ double quote

The XMLDOMEntity object represents an entity in the childNodes property of the docType (DTD) property of the DOMDocument. So, if you want to find out what has been declared as entities in the DTD, just search through the child Nodes of the XMLDOMDocument object.

As mentioned above for the XMLDOMDocumentType object, you cannot add an entity declaration to the DTD in an XML file, as the XMLDOMDocumentType object is read-only.

XMLDOMProcessingInstruction

Any processing instruction found in an XML file is returned as an XMLDOMProcessingInstruction object. Therefore, it's good practice to check the type of Node that you are working with, in case one of these PIs or comments pop up. For example, this PI has been added in between an element:

```
<PERSON id="p1">  
  
<?realaudio version="4.0"?>  
  
<NAME>Mark Wilson</NAME>  
  
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>  
  
<TEL>(++612) 12345</TEL>
```

```
<FAX>(++612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

Ø The same issue applies to comments in your XML file. They are represented as Nodes as well.

Ø As the `XMLDOMProcessingInstruction` object is writable, you can add a PI to a `DOMDocument` (see the `nodeType` property later in this chapter).

XMLDOMParseError

If the `DOMDocument` finds any errors when it parses the XML file or string, the `XMLDOMParseError` object returns the last parse error details. The details that are passed are the line number where the error occurred, the character position on the line, the reason, etc.

The `XMLDOMParseError` object is the only object that the `DOMDocument` inherits. Therefore, you never really need to access it, because it's part of the `DOMDocument` properties.

Example

The following example checks whether there have been any errors after an XML file has been loaded into a `DOMDocument`:

```
Dim objDOMDocument As DOMDocument

Dim objXMLDOMEElement As IXMLDOMEElement

Set objDOMDocument = New DOMDocument

objDOMDocument.async = False

objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"

If objDOMDocument.parseError.reason <> "" Then

    ' there has been an error with the loaded XML – show the reason

    MsgBox objDOMDocument.parseError.reason

End If
```

XMLHttpRequest

The XMLHttpRequest object provides methods that enable you to establish a connection to a file or an object on a web server, such as an ASP page. The two major methods are open() to make the connection and send() to send your request. There are also properties that can be used to read the response: responseText, responseStream, responseXML, and responseBody.

Example

The following example makes a connection with an ASP page. The web page then prepares the XML and returns it. The XMLHttpRequest property called responseXML can be used to access that XML.

```
Dim objXMLHttp As New XMLHttpRequest  
  
Dim objDOMDocument As DOMDocument  
  
objXMLHttp.Open "POST", "http://localhost/xmlcode/demo.asp", False  
  
objXMLHttp.Send  
  
Set objDOMDocument= objXMLHttp.responseXML
```

- Ø Open the POST (or GET) connection to the web server.
- Ø Establish the connection.
- Ø Receive the response.

DOM object properties

These are some of the properties we think would be useful in the context of this book. The descriptions below are limited to our needs and, while being detailed they may not fully describe the capabilities of each object. More complete descriptions can be read from the documentation or on the Microsoft website.

The listing below is a list of all the properties for all of the XMLDom objects. Use this chart when you need to find a property in one of the interfaces that could be useful for something that you are trying accomplish. The properties that are bolded in table are the ones that we will explain. As mentioned before, they have been selected because they are properties and methods we think would be useful in the context of this book.

DOMDocument properties (continued)

	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	O	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	T	M

preserve WhiteSpace	Y																		
previous-Sibling	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
publicId										Y			Y						
readyState	Y																		Y
reason													Y						
resolve-Externals	Y																		
responseBody																			Y
response-Stream																			Y
responseText																			Y
responseXML																			Y
specified	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
srcText													Y						
status																			Y
statusText																			Y
systemId										Y			Y						
tagName									Y										
target																	Y		
text	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
url	Y												Y						
validateOnParse	Y																		
value					Y														
xml	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	

We will use the People2.xml file example for the rest of this chapter. This file's source is:

```
<?xml version="1.0"?>
<!-- ***** Resumes for People ***** -->
<!DOCTYPE PEOPLE SYSTEM "http://localhost/xmlcode/people.dtd">
<PEOPLE>
  <PERSON PERSONID="p1">
    <NAME>Mark Wilson</NAME>
    <ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
    <TEL>(++612) 12345</TEL>
    <FAX>(++612) 12345</FAX>
```

<EMAIL>markwilson@somewhere.com</EMAIL>

</PERSON>

<PERSON PERSONID="p2">

<NAME>Tracey Wilson</NAME>

<ADDRESS>121 Zootle Road, Cape Town, South Africa</ADDRESS>

<TEL>(+2721) 531 9090</TEL>

<FAX>(+2721) 531 9090</FAX>

<EMAIL>Tracey Wilson@somewhere.com</EMAIL>

</PERSON>

<PERSON PERSONID="p3">

<NAME>Jodie Foster</NAME>

<ADDRESS>30 Animal Road, New York, USA</ADDRESS>

<TEL>(+1) 3000 12345</TEL>

<FAX>(+1) 3000 12345</FAX>

<EMAIL>Jodie Foster@somewhere.com</EMAIL>

</PERSON>

<PERSON PERSONID="p4">

<NAME>Lorrin Maughan</NAME>

<ADDRESS>1143 Winners Lane, London, &UK;</ADDRESS>

<TEL>(+94) 17 12345</TEL>

<FAX>+94) 17 12345</FAX>

<EMAIL>Lorrin Maughan@somewhere.com</EMAIL>

</PERSON>

```
<PERSON PERSONID="p5">
  <NAME>Steve Rachel</NAME>
  <ADDRESS>90210 Beverly Hills, California, &USA;</ADDRESS>
  <TEL>(++1) 2000 12345</TEL>
  <FAX>(++1) 2000 12345</FAX>
  <EMAIL>Steve Rachel@somewhere.com</EMAIL>
</PERSON>
</PEOPLE>
```

async

Is a member of: DOMDocument

Syntax

```
blnValue = objDOMDocument.async
```

```
objDOMDocument.async = blnValue
```

Remark

The `async` property indicates or sets a boolean as to whether the XML document is downloaded asynchronously or synchronously. This should be set before loading the XML document, if you do not want to use the default.

The default for this property is set to `True`. This default is important to remember when dealing with the `load()` method of the `DOMDocument`, as it will load your information asynchronously. Therefore, as your `DOMDocument` is busy loading the XML, your VB code will continue to run without waiting for the documents to finish loading.

If you have set your `DOMDocument` to run asynchronously (or have forgotten to change it to *false*), it will then proceed to call another method afterwards, like:

```
Set objPeopleRoot = m_objDOMPeople.documentElement
```

This will cause an *object has not been set* error. This happens because the `DOMDocument` has not yet completed loading, and you are trying to access it. Therefore, before calling the `load()` method, don't forget to set the `async` property to `False`, if you want your XML document to be loaded synchronously.

Example

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMElement As IXMLDOMElement
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

Ø The other approach is to use the `WithEvents` language statement in order to expose the event that come with the `DOMDocument`. Within these events, you can catch when the document has completed loading.

Ø For a more detailed example of using `WithEvents` and the `async` method, see the section on Loading a file asynchronously in chapter 4, “Programming with XML.”

Attributes

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText

XMLRuntime

Syntax

```
set objXMLDOMNamedNodeMap = objDOMDocument.attributes
```

Remark

For more information on attributes, see chapter 2, “XML boot camp.”

This property returns an XMLDOMNamedNodeMap object, which contains a collection of the attributes for this Node. The XMLDOMNamedNodeMap is an XML interface specifically designed for working with attributes.

Example

A common practice is to use attributes to store the ID (using the ID attribute type) from a database, which is declared in the DTD as:

```
<!ELEMENT PERSON ( NAME, ADDRESS, TEL, FAX, EMAIL ) >
```

```
<!ATTLIST PERSON PERSONID ID #REQUIRED>
```

Ø Declaring our Person element will contain the following elements (tags). The person element has an ID attribute called “PERSONID,” which is always required.

Ø In our XML example, this attribute is used as follows:

```
<PERSON PERSONID="p1">
```

```
<NAME>Mark Wilson</NAME>
```

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

```
<TEL>(+612) 12345</TEL>
```

```
<FAX>(+612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

For this element, its ID attribute equals “p1.”

Ø In the following example in VB, we are busy populating the TreeView with data from a XmlDocument. We then want to store the PERSONID attribute in the tag property of the TreeView Node:

'the element Node is holding the id attribute that we want

'to store in the tag, as an identity reference. We

'therefore need to get hold of this Node to get its value

Set objAttributes = objNode.Attributes

'check that there are attributes.

If objAttributes.length > 0 Then

'we know that we've named our id reference as

"PERSONID", therefore tell the NameNodeListMap to get

'this Node by using the getNamedItem method

Set objAttributeNode = objAttributes.getNamedItem("PERSONID")

'store this value in the tag of the TreeView

tvwElement.Tag = objAttributeNode.nodeValue

End If

Ø Get the attributes from the current Node (PERSON Node).

Ø getNamedItem only returns a single Node (IXMLDOMNode), because we have to specify that it must return the "PERSONID" attribute, of which there can only be one per element.

Ø Read the sections on getNamedItem(), setNamedItem(), and nextNode() further in this chapter to learn more about working with attributes in the DOMDocument.

childNodes

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment

XMLDOMDocumentFragment

XMLDOMDocumentType

XMLDOMEElement

XMLDOMEntity

XMLDOMEntityReference

XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XMLRuntime

Syntax

```
set objXMLDOMNodeList = objXMLDOMNode.childNodes
```

Remark

This property is read-only. The `childNodes` property returns the `IXMLDOMNodeList` collection object of the child Nodes for the parent object. Each Node that is returned from the child Node, may have `childNodes` themselves, which is consistent with the tree metaphor that the `DOMDocument` uses.

Example

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNodeList As IXMLDOMNodeList
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMNodeList = objDOMDocument.documentElement.childNodes
```

Ø For the following explanations, we will use this XML example:

```
<?xml version="1.0" ?>
```

```
<!-- ***** Resumes for People ***** --->
```

```
<!DOCTYPE PEOPLE SYSTEM "http://localhost/xmlcode/people.dtd">
```

```
<PEOPLE>
```

```
<PERSON id="1">
```

```
<NAME>Mark Wilson</NAME>
```

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

```
<TEL>(++612) 12345</TEL>
```

```
<FAX>(++612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

```
<PERSON id="2">
```

```
<NAME>Tracey Wilson</NAME>
```

```
<ADDRESS>121 Zootle Road, Cape Town, South Africa</ADDRESS>
```

```
<TEL>(++2721) 531 9090</TEL>
```

```
<FAX>(++2721) 531 9090</FAX>
```

```
<EMAIL>Tracey Wilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

```
</PEOPLE>
```

Common uses of the `childNodes` property are:

- Ø the child Node of the `DOMDocument` root consists of the processing instructions, DTDs, etc.
- Ø the child Node of the root element consists of all the data in the `DOMDocument`
- Ø the child Node of the `attributes` property of an element
- Ø consists of the collection of attributes for the given element

What's so good about the childNodes property?

Ø Because the childNodes property is an XMLDOMNodeList collection, it is easy to iterate through, to collect the data.

Ø You can use the XMLDOMNode object to iterate through the XMLDOMNodeList collection, as it receives most of the other type of interfaces. See later in this section, "Iterating through the childNodes."

Getting the childNodes from the root of the DOMDocument

Here we look for the child Nodes of the actual root of the complete DOMDocument. To get these child Nodes from VB, our code would read:

```
set objXMLDOMNodeList = objDOMDocument.childNodes
```

Ø This will return the four child Nodes shown in figure .

Ø You might have noticed that we have specified the NodeTypes returned from each child Node. Please see the nodeType property for more details on these types of Nodes.

Getting the childNodes from the documentElement element of the DOMDocument

This property returns all the Nodes associated with root Node, which is found in the documentElement property of the DOMDocument (i.e., all the PERSON's in the PEOPLE element). To get the childNodes from VB, our code would read:

```
set objXMLDOMNodeList = objDOMDocument.documentElement.childNodes
```

From our example, two child Nodes will be returned in this NodeList object, as shown in figure .

Getting all the childNodes from a single element

We now want to return all the elements between each section:

```
<PERSON PERSONID="p2">
```

...

```
</PERSON
```

This returns the NAME, ADDRESS, TEL, etc. This is the base of the child Nodes, but even these childNodes can once again iterate down to more childNodes, if they themselves have children.

In the following XML example, the child Node of Mark Wilson is Male:

```
<PERSON PERSONID="p2">
```

```
<NAME>Mark Wilson</NAME>
```

```
<GENDER>Male</GENDER>
```

```
<ADDRESS>
```

```
...
```

```
</PERSON
```

In our main example in table , there are five child Nodes found for each Person element, as shown in figure .

Iterating through the childNodes

Because an `IXMLDOMNodeList` collection is returned from the `childNodes` property, you can iterate through each item using a `for each ... next` loop. Here we demonstrate looping through the collection of child Nodes returned from the `documentElement` property:

```
For Each objNode In objDOMDocument.documentElement.childNodes
```

```
    populateTreeWithChildren objNodeNext
```

The `objNode` has been declared as an `IXMLDOMElement`.

Ø Note that an `XMLDOMNodeList` object is returned even if there are no children of the Node. In such a case, the length of the list will be set to zero. To get the length of the `XMLDOMNodeList`, use:

```
if objPeopleRoot.length > 0 then
```

```
    ...
```

```
end if
```

data

Is a member of: `XMLDOMCDATASection`

`XMLDOMComment`

`XMLDOMProcessingInstruction`

`XMLDOMText`

Syntax

```
strValue = objXMLDOMComment.data
```

```
objXMLDOMComment.data = strValue
```

Remark

This property is readable and writable. Depending on the type of Node, it returns the same value as the nodeValue property. Please see the nodeValue property for more details.

Example

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNode As IXMLDOMNode
```

```
Dim objXMLDOMPI As IXMLDOMProcessingInstruction
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
For Each objXMLDOMNode In objDOMDocument.childNodes
```

```
    If objXMLDOMNode.nodeType = NODE_PROCESSING_INSTRUCTION Then
```

```
        Set objXMLDOMPI = objXMLDOMNode
```

```
        MsgBox "The PI is : " & objXMLDOMPI.Data
```

```
    End If
```

```
Next
```

Ø Check that the nodeType is a Processing Instruction.

Ø Cast the PI object from the Node object.

Ø Retrieve the PI's text using the data property, which from our XML example, returns version="1.0."

docType

Is a member of: XMLDOMDocument

Syntax

```
set objXMLDOMNode = objXMLDOMNode.docType
```

```
set objXMLDOMDocumentType = objXMLDOMNode.docType
```

Remark

This property is read-only.

The doctype is the DTD Node in the XML header. One problem that you may come across when using the DOMDocument is that you might like to add a DTD to your DOMDocument. However, because this property is read-only, you can't add a DTD and an error will occur.

So how does one get around adding a DTD to an XML file? You may think that you can just populate a string that will build up an XML file, which you can do. However, please read about the DOMDocument bug under the loadXML() method, which also explains a solution to this problem.

Example

The following XML code defines a DTD for the XML file:

```
<!DOCTYPE PEOPLE SYSTEM "http://localhost/xmlcode/people.dtd">
```

This VB example shows that a DocumentType object or a Node object is returned from the docType property of the DOMDocument root:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMDocumentType As IXMLDOMDocumentType
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.resolveExternals = True
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
Set objXMLDOMDocumentType = objDOMDocument.doctype
```

Dual interfaces

What you may also have noticed in the properties syntax code example above is that the docType property is being used to return values to two different objects. Take another look:

```
set objXMLDOMNode = objXMLDOMNode.docType
```

```
set objXMLDOMDocumentType = objXMLDOMNode.docType
```

∅ The values are being set, which means an actual object is being passed.

If objectA and objectB are two totally different objects, how then can you set objA = objC and *also* set objB = objC? In other words, how can you place objC into the two *fundamentally different* objects of objA and objB?

This is one of our first examples of an object being able to return two inter-faces. The docType property evidently implements two different interfaces. One of these is applicable to objXMLDOMNode, and the other to objXMLDOMDocumentType. This is because the XMLDOMDocumentType object inherits the objXMLDOMNode object. We can have a closer look at this.

Let's have a look at the *Local View* window in Visual Basic. This screen appears under the Tools/Local View menu and shows the local variable details. Under the Type section, you can see that this object returns the IXMLDOMDocumentType inter-face and the IXMLDOMNode interface. Therefore, you can use either interface to read this Node, depending on the type of interface you choose to use (whichever prop-erties/methods you need to work with).

However, if you look in Visual Basic's Object Browser (press the F2 key to see the object browser), you will find that this object only returns the IXMLDOMDocument-Type. Figure shows the IXMLDOMDocumentType interface in the Object Browser.

While we talk about the docType property, the attributes property of IXM-DOMDocument and IXMLDOMNode returns the actual URI of the DTD.

If you look closer at the docTypes child Node property, you will find a collec-tion of the entities for the DTD.

documentElement

Is a member of: XMLDOMDocument

Syntax

```
set objXMLDOMEElement = objDOMDocument.documentElement
```

```
set objXMLDOMNode = objDOMDocument.documentElement
```

```
set objDOMDocument.documentElement = objXMLDOMEElement
```

```
set objDOMDocument.documentElement = objXMLDOMNode
```

Remark

This property is read-only.

Objects, objects, objects!! If you just want to get to the nitty-gritty and find the XML data in an XML document, this is the property for you! This is the *root element* of your XML (not the root).

Its child Node collection returns all the elements, which we have explained in detail for the childNodes property.

Once again, this property returns two interface types, namely IXMLDOMElement and IXMLDOMNode. As you read more about the other properties and methods later in this chapter, you can decide which interface you prefer to work with when view-ing and manipulating your XML data.

If there is no root element, then Null is returned.

Example

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMElement As IXMLDOMElement
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMElement = objDOMDocument.documentElement
```

firstChild, lastChild, nextSibling, and previousSibling

Is a member of: DOMDocument

XMLDOMNode

XMLDOMAttribute

XMLDOMCDATASection

XMLDOMComment

XMLDOMDocumentFragment

XMLDOMDocumentType

XMLDOMElement

XMLDOMEntity

XMLDOMEntityReference

XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
set objXMLDOMNode = objDOMDocument.firstChild
```

```
set objXMLDOMNode = objDOMDocument.lastChild
```

```
set objXMLDOMNode = objDOMDocument.nextSibling
```

```
set objXMLDOMNode = objDOMDocument.previousSibling
```

Remark

These properties are read-only.

These properties allow us to read the values of the first Node, last Node, etc., from the current Node.

Example

Perhaps one only wants to get the first child of all the elements for an XML file. Therefore, in the following example, we know that we want to populate a text box with the Name and Address from our People2.xml example.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNodeList As IXMLDOMNodeList
```

```
Dim objXMLDOMNode As IXMLDOMNode
```

```
Dim strInfo As String
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"

Set objXMLDOMNodeList = objDOMDocument.documentElement.childNodes

For Each objXMLDOMNode In objXMLDOMNodeList

    strInfo = strInfo & objXMLDOMNode.firstChild.text & vbCrLf

    strInfo = strInfo & objXMLDOMNode.nextSibling.text & vbCrLf & vbCrLf

Next

txtInfo.text = strInfo
```

- Ø Iterate through the collection of elements in the NodeList object.
- Ø The firstChild we know is going to return the Name element.
- Ø The nextSibling returns the address element.

length

Is a member of:

- XMLDOMNodeList
- XMLDOMNamedNodeMap
- XMLDOMComment
- XMLDOMText

Syntax

```
lngValue = objXMLDOMNodeList.length
```

Remark

This property is read-only.

Depending on the DOM object, the length property is used as follows:

For the XMLDOMNodeList and XMLDOMNamedNodeMap interfaces, the length property specifies the number of child Nodes in its collection. We use this property often to verify that an object has been loaded properly, by checking if the length property of the object is greater than zero.

For the XMLDOMComment and XMLDOMText interfaces, the length property returns the number of characters in the body of the Node.

Example

In this example, we test the length to see if a DOMDocument has loaded prop-erly. If the length is greater than zero, then we can proceed with the rest of the code for working with the DOMDocument object.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNode As IXMLDOMNode
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
If objDOMDocument.childNodes.length > 0 Then
```

```
    'run rest of code here
```

```
End If
```

Ø Our code has loaded the XML from our main example. The length returns 4, as there are 4 childNodes for this DOMDocument.

namespaceURI

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference

XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
strValue = objDOMDocument.namespaceURI
```

Remark

This property is read-only. If a namespace has been specified in the XML document, this property returns a string of the location of the URI. As this property is read-only, you cannot add a namespaceURI to a DOMDocument.

Example

In the following example, we checking if the namespaceURI has been set for the DOMDocument.

```
Dim objDOMDocument As DOMDocument
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
If objDOMDocument.namespaceURI <> "" then
```

```
    'run some code here ...
```

```
End If
```

Ø No namespaceURI has been set for this DOMDocument.

nodeName

Is a member of: DOMDocument

XMLDOMNode

XMLDOMAttribute

XMLDOMCDATASection

XMLDOMComment
XMLDOMDocumentFragment
XMLDOMDocumentType
XMLDOMEElement
XMLDOMEntity
XMLDOMEntityReference
XMLDOMNotation
XMLDOMProcessingInstruction
XMLDOMText
XTLRuntime

Syntax

```
strValue = objDOMDocument.nodeName
```

Remark

This property is read-only. Depending on the type of Node, this property returns a string containing the Node's name. This is especially useful with elements and attributes. Table details what is being represented by the nodeName property for the different interfaces of the DOMDocument.

Return values for the different XMLDOM interfaces (continued)

Interface name	Return value
XMLDOMEElement	<p>The name of the element tag is returned, without the tag indicators (</>).</p> <p>XML Example: <EMAIL>markwilson@somewhere.com</EMAIL></p> <p>Returns: EMAIL</p>

	<p>When dealing with namespaces, the full tag name is returned.</p> <p>XML Example: <resume:EMAIL>markwilson@some where.com</resume:EMAIL></p> <p>Returns: resume:EMAIL</p>
XMLDOMAttribute	<p>The name of the element is returned.</p> <p>XML Example: <PERSON PERSONID="p1"></p> <p>Returns: PERSONID</p>
XMLDOMProcessingIn struction	<p>The target of the processing instruction is returned. This is the first word after the '<?' indicators.</p> <p>XML Example: <?xml version="1.0" ?></p> <p>Returns: xml</p>
XMLDOMEntityReference	<p>The name of the entity that is being references is returned, stripping off the entity reference indicators, namely the "&:".</p> <p>XML Example: <ADDRESS>30 Animal Road, New York, &USA;</ADDRESS></p> <p>Returns: USA</p>
XMLDOMEntity	<p>The name of the entity is returned.</p> <p>XML Example: <!ENTITY USA "United States of America"></p> <p>Returns: USA</p>
XMLDOMDocumentType	<p>The name of the document type (DTD) is returned.</p> <p>XML Example: <!DOCTYPE PEOPLE SYSTEM "http://localhost/xml code/people.dtd"></p> <p>Returns: PEOPLE</p>

XMLDOMNotation	The name of the notation is returned. XML Example: Returns: img
-----------------------	---

The Node types listed in table do not have nodeName; therefore, they return the following string literals of what is being referenced.

Return values for XMLDOM interfaces with no nodeName

Interface name	Return value
XMLDOMText	#text
XMLDOMComment	#comment
XMLDOMCDATASection	#cdata-section
XMLDOMDocument	#document
XMLDOMDocumentFragment	#document-fragment

Example

When we want to populate the textboxes on a VB form, we use the nodeName property to work with a Select Case as the decider of which text box should be populated.

In the following example, we have already obtained a reference to an element, which in this case is the PERSON element from our main XML example. We are now looping through its child Nodes, differentiating which Node is current by its nodeName property.

```
For Each objChildElement In objPersonElement.childNodes
```

```
    If objChildElement.nodeType = NODE_ELEMENT Then
```

```
        Select Case UCase(objChildElement.nodeName)
```

```
            Case "NAME"
```

```
                txtName.Text = objChildElement.nodeTypedValue
```

```
            Case "ADDRESS"
```

```
txtAddress.Text = objChildElement.nodeTypeValue
```

```
Case "TEL"
```

```
txtTel.Text = objChildElement.nodeTypeValue
```

```
Case "FAX"
```

```
txtFax.Text = objChildElement.nodeTypeValue
```

```
Case "EMAIL"
```

```
txtEmail.Text = objChildElement.nodeTypeValue
```

```
End Select
```

```
End If
```

```
Next objChildElement
```

nodeType

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
objDOMNodeTypeEnum = objDOMDocument.nodeType
```

```
lngType = objDOMDocument.nodeType
```

Remark

This property is read-only.

Each Node will have a nodeType enumeration (enum) property to distinguish what type of Node we are currently working with. Knowing the Node type also determines whether the Node will have child Nodes.

Example

In each section, we mention what child Node types each Node type can have. Each Node type has its own interface; for example, NODE_ELEMENT uses the XML-DOMElement. However, as mentioned earlier, we will not discuss each one of these interfaces.

In the following example, we iterate through the child Nodes of the DOMDocument. Depending on what type of Node it is, using the nodeType property, we display a message box of what its type is.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNode as IXMLDOMNode
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load http://localhost/xmlcode/people2.dtd
```

```
If objDOMDocument.length > 0 then
```

```
For Each objNode In objDOMDocument.childNodes
```

```
    Select Case objNode.nodeType
```

```
        Case NODE_DOCUMENT_TYPE
```

```
            MsgBox "This is the dtd"
```

```
        Case NODE_ELEMENT
```

```
MsgBox "This is an element"
```

```
Case NODE_PROCESSING_INSTRUCTION
```

```
MsgBox "This is a processing instruction"
```

```
End Select
```

```
Next objNode
```

```
End If
```

You may have noticed that we used an `IXMLDOMNode` object (`objNode`) for iterating through the collection of child Nodes in the `DOMDocument`. This is because most of the XML objects inherit the `IXMLDOMNode` interface. Due to this, you can cast the object into the preferred interface once you have assessed what type of Node you are dealing with.

For example, if your `nodeType` for your object is `NODE_DOCUMENT_TYPE`, then you can cast it like this:

```
set objXMLDOMDocumentType = objNode
```

Now you can work with a declared `IXMLDOMDocumentType` object and use its properties and methods. Let's look more closely at the different `nodeTypes`.

The 12 Node types

Table lists the 12 Node types.

The 12 Node types

Name	Enumeration number
<code>NODE_ELEMENT</code>	(1)
<code>NODE_ATTRIBUTE</code>	(2)
<code>NODE_TEXT</code>	(3)
<code>NODE_CDATA_SECTION</code>	(4)
<code>NODE_ENTITY_REFERENCE</code>	(5)
<code>NODE_ENTITY</code>	(6)
<code>NODE_PROCESSING_INSTRUCTION</code>	(7)
<code>NODE_COMMENT</code>	(8)
<code>NODE_DOCUMENT</code>	(9)
<code>NODE_DOCUMENT_TYPE</code>	(10)
<code>NODE_DOCUMENT_FRAGMENT</code>	(11)

NODE_NOTATION	(12)
----------------------	------

Now let's look at each type in a bit more detail.

NODE_ELEMENT

Has an interface type of: `IXMLDOMElement`
Can have the following children types:
`Element`
`Text`
`Comment`
`ProcessingInstruction`
`CDATASection`
`EntityReference`

This nodeType specifies an element Node in the XML file.

Here is an example of an element in the XML code:

```
<PEOPLE> ... </PEOPLE>
```

This is also an example of an element in the XML code:

```
<TEL>(++) 61 2 12345</TEL>
```

The following XML code includes all the elements you can find in our DOMDocument:

```
<PEOPLE>  
<PERSON id="p1">  
  <NAME>Mark Wilson</NAME>  
  <ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>  
  <TEL>(++612) 12345</TEL>  
  <FAX>(++612) 12345</FAX>  
  <EMAIL>markwilson@somewhere.com</EMAIL>  
</PERSON>
```

...

</PEOPLE>

- ∅ This element is returned by the documentElement property.
- ∅ This element is a childNode of PEOPLE.
- ∅ This element is the first childNode of PERSON.
- ∅ Second childNode of PERSON.
- ∅ Third childNode of PERSON.
- ∅ Fourth childNode of PERSON.
- ∅ Fifth childNode of PERSON.

NODE_ATTRIBUTE

Has an interface type of: XMLDOMAttribute

Can have the following children types: Text

EntityReference

This nodeType specifies an attribute Node in the XML file. For more information on attributes, see the attributes property of this section, as well as chapter 4, “Programming with XML.”

Our XML will read as follows:

```
<PERSON PERSONID="1">
```

PERSONID="1" is the attribute.

In order to work with the XML file, these attributes need to be declared in the DTD file or a section of the XML file, which looks like:

```
<!ATTLIST PERSON PERSONID ID #REQUIRED>
```

In this example, an “id” type attribute declaration, which must exist for each Person element in the XML file as an id type attribute, is declared as “#REQUIRED.”

NODE_TEXT

Has an interface type of: XMLDOMText

Can have the following children types: None

This nodeType specifies a text Node in the XML file. A text Node can appear as the child Node of Attribute, DocumentFragment, Element, and EntityReference Nodes. It never has any child Nodes.

In the following XML example, the highlighted text will be the value in a text Node:

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

The highlighted text is the Node text.

If you are working with an ElementNode type, then you actually don't need to iterate down to its child Node to get the text of the element. You can just use the node-TypeValue property of the element Node. This will give you its text value, whether it contains a CDATA section, entities, or whatever.

However, there are a few tricks. The next Node type we will explain is the CDATA section. If your Node happens to be of this type, it will be represented as a CDATA section Node and not a text Node, even though it looks like a text Node. So, if you are looking for the text Node of an element Node, don't forget to look for the CDATA section Node as well.

NODE_CDATA_SECTION

Has an interface type of: XMLDOMCDATASection

Can have the following children None

types:

Sometimes you will want to insert unusual characters in your XML Nodes. To do this, you need to use a CDATA section (also known as a Marked section).

Yes, we know that you can also use built-in entities instead of a CDATA section, but there are currently only five of these built-in entities that the DOMDocument will parse. (See the XMLDOMEntity object section earlier in this chapter.)

But built-in entities don't sort out all our problems. What do we do about a hash character (#)? An XML parser does not allow you to use a # character in your XML, but there is no DOMDocument built-in entity for a # character.

Note

Built-in entities are like VB constants, but they are built into XML. Because XML has certain characters that are used specifically for XML (like a < character), you can use the entity to represent your character.

Here we talk specifically about what the DOM-Document can parse. Many browsers support numerous entities that they recognize, but the DOMDocument seems to support only these five built-in entities.

To use a CDATA section this in our XML code, instead of declaring an element in the DTD as CDATA element, we can use the following in our XML code:

```
![CDATA[]]
```

Here is an example of an element that uses a CDATA section with a # in it:

```
<ADDRESS><![CDATA[#911 Somewhere Circle, Canberra, Australia]]></ADDRESS>
```

Note

If you do a Select Case in VB to find the enumeration of a Node, using the NODE_TEXT you will miss the text that is put into CDATA sections to protect reserved characters. Therefore, don't forget to look for NODE_CDATA_SECTION in your Select Case.

A CDATA section Node can be the child Node of Elements, EntityReferences, and DocumentFragments.

If you try to use an Entity in a CDATA section, the entity will not be parsed, and therefore not interpreted. For example the following XML code is embedded in a CDATA section:

```
<ADDRESS><![CDATA[#911 O&apos;Hara Circle, Canberra, Australia]]></ADDRESS>
```

Did you note the **'** entity? The **'** entity represents an ' (apostro-phe) and we would expect the output to display *O' Hara Circle*, wouldn't we? How-ever, this will not be interpreted as O'Hara Circle, because the text is contained in CDATA section and won't be parsed. When we examine the above XML code from VB, using the nodeTypedValue property to get the value of the element, we observe the following:

```
strValue = objXMLElement.nodeTypedValue
```

strValue returns: #911 O'Hara Circle, Canberra, Australia.

Note

When you allow users to add data to a DOMDocument, don't forget to check for any unparsable characters. Embed these unparsable characters in a CDATA section before saving the DOMDocument. If you do not this, when someone has typed in an apostrophe in one of the fields (e.g., O'Mal-ly), the DOMDocument will save this and return no errors. However, when you try to load this saved XML file into the DOMDocument, you will re-ceive an error because of this unparsable character.

NODE_ENTITY_REFERENCE

Has an interface type of: XMLDOMEntityReference

Can have the following children Element

types:

ProcessingInstruction

Comment

Text

CDATASection

EntityReference

This Node specifies a reference to an entity (see the next section on the Entity Node).

Entities are specified in the DTD.

In the DTD file, we specify the entity as:

```
<!ENTITY USA "United States of America">
```

In the XML file, the following code in an element Node will have two child Nodes, one a text Node and the other an EntityReference:

```
<ADDRESS>30 Animal Road, New York, &USA;</ADDRESS>
```

In the DOMDocument, the XML example above can be extracted, for the element, as follows:

First child Node

```
strValue = objXMLElement.childNodes(1).nodeValue
```

```
enumType = objXMLElement.childNodes(1).nodeType
```

Returns: strValue = "30 Animal Road, New York."

Returns: enumType = NODE_TEXT.

Second child Node

```
strValue = objXMLElement.childNodes(2).Text
```

```
enumType = objXMLElement.childNodes(2).nodeType
```

Returns: strValue = "United States of America."

Returns: enumType = NODE_ENTITY_REFERENCE.

However, why have we changed from using the `nodeValue` property in the first child to using the `Text` value in the second child? It's because we are dealing with two different Node types. Therefore, depending on your Node type, you get the data you need to access different properties. See the `nodeValue` and `Text` in this section for more information on this.

Note

If you only want to get the value of the element Node, you don't have to go through this long procedure to get the value. You only have to do this if you specifically want to analyze the EntityReference Node, for example.

The following XML code specifies an entity “&USA;”:

```
<ADDRESS>30 Animal Road, New York, &USA;</ADDRESS>
```

In the DOMDocument, you can get straight to finding its parsed value from the element Node:

```
strValue = objXMLElement.nodeTypeValue
```

Returns: strValue = “30 Animal Road, New York, United States of America.”

NODE_ENTITY

Has an interface type of: XMLDOMEntity

Can have the following Text

children types:

EntityReference

As mentioned earlier, an entity can only be *specified* in the DTD section or file, and not in the XML file. Therefore, entities will always be child Nodes of the DocumentType Node (see the docType property in this section).

An example of entities in the DTD is:

```
<!ENTITY USA "United States of America">
```

```
<!ENTITY UK "United Kingdom">
```

As you can see in figure , there are two child Nodes (Item 1 and Item 2) for the docType property of this object.

From our DTD sample code above, the DOMDocument’s docType child– Nodes property has two child Nodes. Our nodeName property for the first child Node of nodeType NODE_ENTITY is USA. This Node consists of one child Node, which is of the type NODE_TEXT. However, as you might have noticed, you can reference the text property—this will give you the same value as the child Node, without having to iterate to the child Nodes.

Although they are referenced (NODE_ENTITY_REFERENCE Node types) later in the XML file, among the elements, the parser will automatically handle expanding these entities—unless it’s in a CDATA section, which we have explained in the NODE_CDATA_SECTION in this section.

NODE_PROCESSING_INSTRUCTION

Has an interface of: XMLDOMProcessingInstructor

Can have the following children None

types:

This type of Node specifies a PI in the XML document. Processing Instructions are declared in the XML file as:

```
<?xml version="1.0" ?>
```

PIs can be found as part of the childNodes property in Document, DocumentFragment, Element, and EntityReference Nodes. Therefore, don't be deceived—they are not only found as the PI at the header of a document.

We could easily put the following XML code, “<?realaudio version="4.0"?>”, among our elements, which would cause no harm to the DOMDocument. This will return six child Nodes for the PERSON element. The realaudio PI is returned as a PI Node (and in case you were wondering, the parser will ignore it completely).

```
<PERSON id="p1">
```

```
<?realaudio version="4.0"?>
```

```
<NAME>Mark Wilson</NAME>
```

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

```
<TEL>(++612) 12345</TEL>
```

```
<FAX>(++612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

NODE_COMMENT

Has an interface type of: XMLDOMComment

Can have the following children None

types:

This Node is a comment. Comments can be inserted anywhere, so this is another Node to be aware of when iterating through a Node collection. It's good practice to check the type of Node that you are working with, in case PIs or comments pop up. The following XML example includes a comment about this person's address that someone inserted.

```
<PERSON id="p1">
```

```
<!-- Person not available for another contract until September -->
```

```
<NAME>Mark Wilson</NAME>
```

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

```
<TEL>(++612) 12345</TEL>
```

```
<FAX>(++612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

In the following VB example, we iterate through our child Nodes for an element. For this example, we only want to check for Element, Comments, or PI Nodes. The `nodeType` property enables us to do this check.

```
For Each objNode In objElement.childNodes
```

```
    Select Case objNode.nodeType
```

```
        Case NODE_COMMENT
```

```
            'Do something to handle comments
```

```
        Case NODE_ELEMENT
```

```
            'Do something to handle elements
```

```
        Case NODE_PROCESSING_INSTRUCTION
```

```
            'Do something to handle PI's
```

```
    End Select
```

```
Next objNode
```

NODE_DOCUMENT

Has an interface type of: XMLDOMDocument

Can have the following children Element

types:

ProcessingInstruction

Comment

DocumentType

This Node type specifies that this Node is the document object, which is the root of the whole document. There can only be one document Node per XML file. In the following example, we load a DOMDocument from an XML file:

```
Dim objDOMDocument As DOMDocument
```

```
Set objDOMDocument = New DOMDocument
```

`objDOMDocument.async = False`

`objDOMDocument.Load http://localhost/xmlcode/people2.dtd`

Ø If all is fine with our XML code, we now have a loaded `DOMDocument`!

In this example, if you look at the `objDOMDocument`'s `nodeType` property, you will see that it's a `NODE_DOCUMENT` type Node.

Of its `childNodes`, the Element Node type can only consist of one element (who in itself has child Nodes), as there is only one root element (see the `documentElement` property for a detailed explanation and example of this Node).

NODE_DOCUMENT_TYPE

Has an interface type of: `XMLDOMDocumentType`

Can have the following children Notation

types:

Entity

This Node type specifies that this is the DTD Node, which is represented in the XML code as:

```
<!DOCTYPE PEOPLE SYSTEM "http://localhost/xmlcode/people.dtd">
```

For a more detailed explanation of the DocumentType Node, see the `docType` property in this section.

NODE_NOTATION

Has an interface type of: `XMLDOMNotation`

Can have the following children None

types:

This is a notation Node. These notations can only be declared in the document type declaration, just like the Entity type Node. Therefore, it will only be found as a child of the Document type Node.

An example of a notation is when we want to tell the browser how to include an image or realaudio or whichever type of file; we can then specify the following:

```

```

nodeTypedValue

Is a member of: `DOMDocument`

`XMLDOMNode`

`XMLDOMAttribute`

XMLDOMCDATASection

XMLDOMComment

XMLDOMDocumentFragment

XMLDOMDocumentType

XMLDOMElement

XMLDOMEntity

XMLDOMEntityReference

XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
vntValue = objXMLDOMElement.nodeTypeValue
```

Remark

In VB's object browser, its description of this property is: property *get the strongly typed value of the node*. For an element Node, it returns a variant datatype of the contents of the Node. If you're working with an attribute Node, you will only get the value of *attribute*, which does not return much value. (See *nodeValue* for working with attribute Nodes.)

If you have used a Schema in your XML, which does not have a datatype specified for the element in the Schema, a string value datatype is the default datatype returned. Otherwise, it returns the data in the specified datatype value. This is the main strength of this property.

When working with an element Node, you could also use the Text property to get the value of a Node. It's a matter of preference.

Example

You cannot use the *nodeTyped-Value* property to set the value of a Node, as it will return a runtime error. You need to use the *text* property to do this.

In the following example, after loading our *DOMDocument*, we want to get the value of the first child's value of the first element (*firstChild*) of the root element (*documentElement*), using the *nodeTypedValue* property:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMELEMENT As IXMLDOMELEMENT
```

```
Dim strFirstValue As String
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMELEMENT = objDOMDocument.documentElement.firstChild
```

```
strFirstValue = objXMLDOMELEMENT.firstChild.nodeTypeValue
```

Ø Get the root Node's first child element.

Ø From the first child of the root Node, get the first child of that element.

Ø This returns "Mark Wilson" from our main XML example.

nodeTypeString

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMELEMENT
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
strValue = objXMLDOMNode.nodeTypeString
```

Remark

The `nodeTypeString` returns the `nodeType` of the current Node in a string. This can be used instead of the `nodeType` property, if you want to work with a string instead of an enumeration datatype. You can use the same examples from the `nodeType` property in this section, except use this property to make it return a string. Table lists the `nodeTypeString` return values.

`nodeTypeString` return values

Type of Node	nodeTypeString value
Element	"element"
Attribute	"attribute"
Text	"text"
CDATA section	"cdatasection"
Entity Reference	"entityreference"
Processing Instruction	"processinginstruction"
Comment	"comment"
Document Fragment	"documentfragment"

nodeValue

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment

XMLDOMDocumentType
XMLDOMEElement
XMLDOMEntity
XMLDOMEntityReference
XMLDOMNotation
XMLDOMProcessingInstruction
XMLDOMText
XTLRuntime

Syntax

vntValue = objXMLDOMAttribute.nodeValue

Remark

This property is readable and writable. The VB object browser specifies that this is the *value stored in the node*. For each nodeType these values can differ as shown in table .

nodeValue returned for nodeTypes (continued)

XMLDOMAttribute

The nodeValue returns a string containing the value of the attribute. If the current attribute has child Nodes, then the string will be a concatenation of all its child Nodes.

XML Example:

```
<PERSON PERSONID="p1">
```

Returns: p1

XMLDOMText

The nodeValue returns a string of the contents of a text Node.

XML Example:

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

Returns: markwilson@somewhere.com

XMLDOMComment

The nodeValue returns a string of the contents of a comment, stripping off the comment indicators and white space.

XMLDOMCDATASection

XML Example:

```
<!-- ***** Resumes for People ***** -->
```

Returns: ***** Resumes for People *****

The nodeValue returns a string of the contents of a CDATASection. As mentioned under the nodeTypes properties, beware of including an entity in a CDATASection, as it does not expand.

In the following example, the hash (#) key is not permitted in XML; therefore, it needs to be embedded in a CDATA section.

XML Example:

```
<ADDRESS><![CDATA[#911 Somewhere Circle, Canberra, Australia]]></ADDRESS>
```

Returns: #911 Somewhere Circle, Canberra, Australia

The nodeValue returns a string of the contents of a processing instruction, stripping off the processing instruction indicators and white space.

XMLDOMProcessingInstruction

XML Example:

```
<?xml version="1.0" ?>
```

Returns: xml version="1.0"

The nodeValue returns Null.

XMLDOMElement

XMLDOMDocument

XMLDOMDocumentType

XMLDOMDocumentFragment

XMLDOMNotation

XMLDOMEntityReference

Example

In the following example, we load a DOMDocument. Then we want to insert the Processing Instruction Node's nodeValue property into a text box (txtPI), after iterating through each Node of the DOMDocument Node collection, until we find the Processing Instruction Node.

```
Dim objDOMDocument As DOMDocument

Dim objNode As IXMLDOMNode

Set objDOMDocument = New DOMDocument

objDOMDocument.async = False

objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"

If objDOMDocument.childNodes.length > 0 Then

For Each objNode In objDOMDocument.childNodes

    Select Case objNode.nodeType

        Case NODE_PROCESSING_INSTRUCTION

            txtPI.Text = objNode.nodeValue

    End Select

Next objNode

End If

Ø    Make sure that there are childNodes after loading the document.

Ø    Check the Node's nodeType enumeration for a PI Node.

Ø    Display the nodeValue of the PI in a text box.
```

ondataavailable

Is a member of: DOMDocument

Syntax

```
strValue = objDOMDocument.ondataavailable
```

Remark

This property is write-only. When an XMLDOMDocument is large, waiting for the XML to load can take quite a while. The DOMDocument provides this prop-erty to let you know when the data has started to become available. You can then start working with this data as it arrives.

This property works in conjunction with the `ontransformnode()` event. For more information on this property, see the “*Loading a file asynchronously*” section in chapter 4, “Programming with XML.”

ownerDocument

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMEElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

```
set objDOMDocument = objXMLDOMNode.ownerDocument
```

Remark

This property is read-only. It returns the parent document to which this Node object belongs. This is always going to be the root of the document (DOMDocument).

parentNode

Is a member of: DOMDocument

XMLDOMNode
XMLDOMAttribute
XMLDOMCDATASection
XMLDOMComment
XMLDOMDocumentFragment
XMLDOMDocumentType
XMLDOMElement
XMLDOMEntity
XMLDOMEntityReference
XMLDOMNotation
XMLDOMProcessingInstruction
XMLDOMText
XTLRuntime

Syntax

```
set objXMLDOMNode = objXMLDOMNode.parentNode
```

Remark

This property is read-only. Whereas the ownerDocument object returns the root of the document, the parentNode returns the parentNode for the current Node. All Nodes except Document, DocumentFragment, and Attribute Nodes can have a parentNode.

When we create a Node and it has not yet been added to the tree—or if it has been removed from the tree—the parent is NULL. (See the createNode() or createElement() methods later in this section for more information on how to do this.)

parsed

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute

XMLDOMCDATASection
XMLDOMComment
XMLDOMDocumentFragment
XMLDOMDocumentType
XMLDOMElement
XMLDOMEntity
XMLDOMEntityReference
XMLDOMNotation
XMLDOMProcessingInstruction
XMLDOMText
XTLRuntime

Syntax

```
blnParsed = objDOMDocument.parsed
```

Remark

This property returns True/False, regarding whether the document is parsed, after it is loaded (i.e., indicates whether it is a well-formed document). It will only parse a document if you have set the resolveExternals property of DOMDocument to True. (See the resolveExternals property for more details.)

Example

The following example is the same as found in the length property example, except we use this property to check whether the XML file is parsed.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNode As IXMLDOMNode
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
If objDOMDocument.parsed = True Then
```

```
'run rest of code here
```

```
End If
```

parseError

Is a member of: XMLDOMDocument

Syntax

```
set objXMLDOMParseError = objDOMDocument.parseError
```

Remark

This property is read-only. This property returns an XMLDOMParseError object. The returned object is always a valid object. When you are dealing with the DOMDocument interface, it inherits the XMLDOMParseError interface; therefore all the properties of the XMLDOMParseError object are exposed in the DOMDocument object (for example, the parseError property). This means you can interrogate any errors immediately after you have loaded your DOMDocument, without having to set an XMLDOMParseError object.

Example

In the following example, we demonstrate using the reason property of the XML-DOMParseError object from the DOMDocument object.

```
If objDOMDocument.parseError.reason <> "" Then
```

```
'there has been an error with the loaded XML – show the reason
```

```
MsgBox objDOMDocument.parseError.reason
```

```
End If
```

However, if you want to interrogate any errors in any of the other types of Nodes, you have to get the XMLDOMParseError object to find any of the error properties:

```
Set objXMLDOMParseError = objXMLDOMElement.parseError
```

```
If objXMLDOMParseError.reason <> "" then
```

```
Msgbox objXMLDOMParseError.reason
```

```
end if
```

prefix

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

```
strVAlue = objXMLDOMNode.prefix
```

Remark

This property is read-only. When working with a namespace in your document, this property returns the prefix for that namespace.

Example

In your XML file:

```
<resume:EMAIL>markwilson@somewhere.com</resume:EMAIL>
```

VB returns:

```
strValue = objNode.prefix
```

strValue returns: resume.

preserveWhiteSpace

Is a member of: XMLDOMDocument

Syntax

```
blnValue = objDOMDocument.preserveWhiteSpace
```

```
objDOMDocument.preserveWhiteSpace = blnValue
```

Remark

This property is a readable and writable property that returns True/False. The default is False. White space is a space, tab, or carriage return (new line) character in your XML file.

This property is useful when you need to preserve the layout (white space) of the data in your XML document. For example, you have as one of your elements a comment that consists of sentences. You would need to preserve the two spaces after a full stop, the carriage return between paragraphs. Well, if you did not specify to preserve the white space for this element, you would lose all this formatting.

In the XML specifications, all white space is meant to be preserved; however, the default behavior of this property in the DOMDocument is False, and as such, the XML and TEXT properties do not preserve the white space.

The text and XML properties will preserve white space when the user has set the preserveWhiteSpace property to True and/or when the xml:space attribute on the XML element has the value *preserve*. Depending on which you choose from the previous line; the DOMDocument object handles the white space differently.

There are different types of white space:

Preserved: the content of the DOMDocument will be exactly as it's found in the XML file

Trimmed: the leading and trailing spaced in your XML file are removed

Half-preserved: the white space inside your text is preserved, but the white space between tags is *normalized*

To find more information about how setting your white space affects the out-put of your text in the DOMDocument, go to the Microsoft website, which explains this quite well.

Example

The following XML example specifies that the white space needs to be preserved:

```
<ADDRESS>xml:space="preserve">
```

911 Somewhere Circle,

Canberra,

Australia

```
</ADDRESS>
```

Now this data will keep its form when it is displayed.

resolveExternals

Is a member of: XMLDOMDocument

Syntax

```
blnVal = objDOMDocument.resolveExternals
```

```
objDOMDocument.resolveExternals = blnVal
```

Remark

This property is readable and writable. True/False is returned/set, defaulting to True. Remember that there is a difference between valid and well-formed. A *merely* well-formed XML document has matching tags and is syntactically correct. Valid means that the entire XML document is correct, including the use of a Schema, DTD, or other externals.

Externals are items that are referenced from within the XML document, such as namespaces, DTDs, and other included files or objects.

When the DOMDocument is parsed, validation can occur or not. We can turn validation off—as is the default for IE5 when it displays XML files—by setting validateOnParse property to False.

During parsing, if validation occurs, validation will fail unless resolveExternals is set to True.

Example

To resolve a namespace, the Microsoft DOM objects require that the URI prefix begins with “x-schema.”

In the following example, the DOM document loaded has a DTD. Before the load takes place, the DOMDocument needs to resolve the DTD. Therefore, the resolveExternals property needs to be set before loading the document. But if we want the XML file validated against the DTD file request, we must set the validateOnParse property to True:

```
Dim objDOMDocument As DOMDocument
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.resolveExternals = True
```

```
objDOMDocument.validateOnParse = True
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

tagName

Is a member of: XMLDOMElement

Syntax

```
strValue objXMLDOMElement.tagName
```

Remark

This property is read-only. For an element, this property returns the tagName of the element. It therefore is the same as the nodeName property, except that it is only found in the XMLDOMElement interface. Please see the nodeName property for a detailed explanation and example.

text

Is a member of: DOMDocument

XMLDOMNode

XMLDOMAttribute

XMLDOMCDATASection

XMLDOMComment

XMLDOMDocumentFragment

XMLDOMDocumentType

XMLDOMElement

XMLDOMEntity

XMLDOMEntityReference

XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
strValue = objXMLDOMNode.text
```

Remark

For writing the values to a Node, the text property is used. The text property normalizes white space, unless the DOMDocument has been specified to preserve the white space (see the preserveWhiteSpace property).

When reading the value of a Node, using the text property can lead to some strange behavior if your current Node has children.

Example

The following XML example explains what we meant by the previous comment.

```
<PEOPLE>
```

```
<PERSON id="1">
```

```
<NAME>Mark Wilson</NAME>
```

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

```
<TEL>(++612) 12345</TEL>
```

```
<FAX>(++612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

```
....
```

```
</PEOPLE>
```

For the other properties, like the nodeType property, we explained how to get references to CDATA sections, etc. However, if you are looking for the text of a Node and are not too concerned about interrogating whether the sub-children consist of CDATA sections or entities, then just getting the text

property will return the parsed data of your element.

In our example we have loaded our DOMDocument, which consists of our People2.xml file. We then get a reference to the firstChild property (objNode = '<PERSON id="1">') of the documentElement property:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNode As IXMLDOMNode
```

```
Dim strText as String
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.resolveExternals = True
```

```
objDOMDocument.validateOnParse = True
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
Set objNode = objDOMDocument.documentElement.firstChild
```

```
strText = objNode.text
```

Our objNode has child Nodes; therefore, its text property returns the values of all its child Nodes. The string strText returns the following:

```
Mark Wilson 911 Somewhere Circle, Canberra, Australia (+612) 12345 (+612) 12345  
markwilson@somewhere.com"
```

The value return has all the tags, and the less than (<) and greater than (>) signs have been removed, with a space inserted between each child for the current Node.

url

Is a member of: XMLDOMDocument

Syntax

```
strValue = objDOMDocument.url
```

Remark

This property is read-only. This property returns the URL for the loaded XML document.

validateOnParse

Is a member of: XMLDOMDocument

Remark

See the resolveExternals property for more information, as these two properties are interlinked.

value

Is a member of: XMLDOMAttribute

Syntax

```
vntValue = objXMLDOMAttribute.value
```

Remark

This property returns the value (content) of an attribute. For the XMLDOMAttribute object, this is the same as the nodeValue property.

Example

In the following example, we first get a reference to the firstChild of the documentElement. We then get a reference to the attribute for the firstChild, and then find the value of that attribute.

```
Dim objDOMDocument As DOMDocument  
  
Dim objNode As IXMLDOMNode  
  
Dim objAttribute As IXMLDOMAttribute  
  
Dim strValue as String  
  
Set objDOMDocument = New DOMDocument  
  
objDOMDocument.async = False  
  
objDOMDocument.resolveExternals = True  
  
objDOMDocument.validateOnParse = True  
  
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"  
  
Set objNode = objDOMDocument.documentElement.firstChild  
  
Set objAttribute = objNode.Attributes(0)
```


appendChild()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

```
set objXMLDOMNode = objDOMDocument.appendChild(objNewChildNode)
```

Remark

This method's function is to append (add) a new child Node to the end of a Node.

Before we can append a new Node (element, attribute, etc.), we need to first create this Node. Then once we have a reference to this Node, we can append it to the current parentNode by passing the parameter (objNewChildNode), which is the newly created Node. This new child will be added to the end of the list of children on this parentNode.

The child (objXMLDOMNode) will be returned if it is successfully added. It will be set to nothing if it fails.

Appending to the root (DOMDocument)

If you recall, we mentioned that the DOMDocument root could only have the following child Nodes:

- Ø element—only one, because the DOMDocument can only have one root element Node (createElement)
- Ø processing instruction (createProcessingInstruction)
- Ø comment (createComment)
- Ø document type (DocumentType)

Of these child Nodes, the only one that you cannot currently add to the DOM-Document is a DocumentType (DTD).

Example

Before we can append a child Node, we need to first create a Node of whichever type you want (attribute, element, etc.) and set a few properties to it if needed. This example creates a new DOMDocument, and then the documentElement gets added to it:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objPeople As IXMLDOMNode
```

```
Dim objNode As IXMLDOMNode
```

```
Set objDOMDocument = new DOMDocument
```

```
Set objPeople = objDOMDocument.createElement("People")
```

```
Set objNode = objDOMDocument.appendChild (objPeople)
```

Ø Add the root Node to the XML document.

Ø Create the element first.

Ø Append this childNode to the end of the DOMDocument.

Consider the following code:

```
Set objNode = objDOMDocument.appendChild (objPeople)
```

Ø You only need to say Set objNode = .

If you want to check that this Node was created properly. You could just say:

```
objDOMDocument.appendChild objPeople
```

Appending to the root element (documentElement)

If you try to add a second element– type Node to the DOMDocument, you’ll get a run– time error—because the DOM-Document only allows one root element.

How do you go about adding more data to your DOMDocument?

No matter how many levels deep your tree goes, you need to add your first branch, which is your documentElement. Thereafter you need to append (add) the rest of the XML data.

The following steps show the sequence for adding data to your XML code, starting with adding the root Node.

Step 1:

```
<PEOPLE>
```

....

```
</PEOPLE>
```

Then you need to add your next level branch, which will be the your first sub-element of the root Node.

Step 2:

```
<PEOPLE>
```

```
  <PERSON>
```

...

```
  </PERSON>
```

```
</PEOPLE>
```

To this subelement we can then add the elements (data).

Step 3:

```
<PEOPLE>
```

```
  <PERSON>
```

```
    <NAME>Mark Wilson</NAME>
```

```
<ADDRESS>911 Somewhere Circle, Canberra, Australia</ADDRESS>
```

```
<TEL>(++612) 12345</TEL>
```

```
<FAX>(++612) 12345</FAX>
```

```
<EMAIL>markwilson@somewhere.com</EMAIL>
```

```
</PERSON>
```

```
</PEOPLE>
```

You now need to append the root element. The following code shows Step 2 and Step 3 added to the XML code in VB. (See createElement() to create the element.)

```
Dim objPerson As IXMLDOMNode
```

```
Dim objChild As IXMLDOMNode
```

```
Set objPerson = objDOMDocument.createElement("PERSON")
```

```
objDOMDocument.documentElement.appendChild objPerson
```

```
Set objChild = objDOMDocument.createElement("NAME")
```

```
objChild.Text = "Monty Python"
```

```
objPerson.appendChild objChild
```

```
objDOMDocument.save
```

Ø Create a new element DOMDocument.

Ø We have already inserted our documentElement. Now we need to append a new Person element to the documentElement.

Ø Create a new childNode for the Person element and give it a few properties.

Ø Append this new child to the Person element.

To actually add any new change to the DOMDocument, you need to save the DOMDocument.

Note

The Node may be added to the instantiated DOMDocument, but it is not added to the actual XML file until you have called the save() method.

In Line 2 above we could also say:

```
objPeople.appendChild objPerson
```

Either method is fine. Both `objPeople` and `objDOMDocument.documentElement` refer to the Document Element.

cloneNode()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

```
set objXMLDOMNode = objXMLDOMNode.cloneNode(deep)
```

Remark

Pass one parameter (`deep`) that is either `True` or `False`. It indicates if the Node should be cloned (copied) with its children included or not. Any changes made to this Node will not be reflected in the `DOMDocument`.

Note that the new Node returned will be of the `XMLDOMNode` interface.

createAttribute(), createCDATASection(), createComment(),createElement(), createEntityReference(),createProcessingInstruction(), createTextNode()

Is a member of: DOMDocument

Syntax

```
objXMLDOMAttribute = objDOMDocument.createAttribute(name)
```

Remark

Adding Nodes to the DOMDocument is done in two phases:

Ø You first need to create a Node of the type you want (Element, Comment, etc.)

Ø Then you append the created Node to the DOMDocument

Creating an element

```
objXMLDOMEElement = objDOMDocument.createElement(tagName)
```

These methods are only applicable to the DOM–Document interface.

When creating an element, we need to pass the createElement() method the tag name of the element to be created. This tag name is case–sensitive. Once we’ve established a new element, then we need to give it its values. See the example below.

In the code for the appendChild() method, we see how an element is created and added to a parentNode. Therefore, we have already created a DOMDocument and its documentElement property (root element). Now we want to add children to the documentElement:

```
Dim objPerson As IXMLDOMNode
```

```
Dim objChild As IXMLDOMNode
```

```
Set objPerson = objDOMDocument.createElement("PERSON")
```

```
objDOMDocument.documentElement.appendChild objPerson
```

```
Set objChild = objDOMDocument.createElement("NAME")
```

```
objChild.Text = "Monty Python"
```

```
objPerson.appendChild objChild
```

Ø The tag name (element name) is passed as a string to say which element to create. We get a reference to

objPerson, because we will add childNodes (Name, Address etc.) to this parentNode.

- Ø Add objPerson Node to the documentElement Node.
- Ø Create another element—note that we are using the DOMDocument object here.
- Ø Add this childNode to the Person element.

Before appending the element to its parentNode, don't forget to give it its properties, like the text, etc.

Here again we see the append method being used to actually add this element to the parentNode. If this is not done, then the new element's parent will be NULL and not part of the DOMDocument.

This create-Element() method does not add the new child-Node to the document tree. To actually add it to the document, you need to call the append-Child() method.

You now need to use a method such as insertBefore(), replaceChild(), or appendChild() to add the new child to the DOMDocument. Then you need to use the save() method on the DOMDocument to actually write the change to the XML file. This applies to the methods that follow as well.

Namespaces

If you have specified a namespace in your document (namespaceURI = ...), the element will be added in the context of this namespace. If no prefix is used before a tagName, then element is added to the default namespace. If you say:

```
Set objChild = objDOMDocument.createElement("resume:NAME")
```

the colon will be ignored.

Creating an attribute

The VB code line:

```
objXMLDOMAttribute = objDOMDocument.createAttribute(name)
```

creates an attribute with a specific name, where name is the name of the attribute. You then need to give the attribute its details. See the example below.

Attributes can only be added to created or current elements. Therefore, you need to first get access to an element object to which you are about to add an attribute.

There are several methods used to add an attribute to an element (which you will see as you go through the methods). If you use the createAttribute() method, this is how to go about it:

```
Dim objNode As IXMLDOMNode
```

```
Dim objAttrib As IXMLDOMAttribute  
Set objNode = m_objDOMPeople.createElement("PERSON")  
Set objAttrib = m_objDOMPeople.createAttribute("PERSONID")  
objAttrib.Text = "p7"  
objNode.Attributes.setNamedItem objAttrib  
m_objDOMPeople.documentElement.appendChild objNode
```

- Ø Create the element Node if it's a new element.
- Ø Create the attribute type, passing it a string of its name.
- Ø Give the attribute properties.
- Ø To the created Node (element), add the attribute object.
- Ø Append the created Node to the documentElement.
- Ø See the setAttribute() method in this chapter.

Creating a CDATA section

The following VB code creates a CDATA section in an element Node in the DOMDocument:

```
objXMLDOMCDATASection = objElement.createCDATASection(data)
```

This method can be really handy if you need to find an easy way to put those crazy brackets (<![CDATA[#]]>) around a CDATA section, without having to put the brackets around yourself when you are busy modifying a DOMDocument.

In the following example, we need to have the following in our address Node:

```
#123 Narrabundah Avenue
```

We accomplish this by wrapping the hash (#) in a CDATA section:

```
Dim objCDATA As IXMLDOMCDATASection  
Set objCDATA = objDOMDocument.createCDATASection("#")  
Set objChild = objDOMDocument.createElement("ADDRESS")
```

```
objChild.Text = objCDATA.XML & "123 Narrabundah Avenue"
```

```
Set objNode = objDOMDocument.documentElement.appendChild(objChild)
```

If you query the text property for this Node (objNode), it returns:

```
<![CDATA [#]]>123 Narrabundah Avenue
```

Creating a comment

The following VB code creates a comment in the DOMDocument:

```
objXMLDOMComment = objDOMDocument.createComment(data)
```

If you need to add a comment to your XML, then you need to use the following method. It automatically adds the correct characters (<!-- -->) around the comment; you only need to add the data.

In the following example, we add a comment to the DOMDocument:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objComment As IXMLDOMComment
```

```
Set objDOMDocument = New DOMDocument
```

```
Set objComment = objDOMDocument.createComment("This is a comment.")
```

```
objDOMDocument.appendChild objComment
```

```
objDOMDocument.save http://localhost/xmlcode/people2.xml
```

Ø Create your DOMDocument.

Ø Create a Comment object, passing just your comment, no XML tags. Note that this object can also be an XMLDOMNode object.

Ø Add this new comment object to the DOMDocument.

If you run this example and look at the xml property for the objComment object, you will see that the comment tags (<!-- -->) have been automatically added, but the text property does not have these tags.

Creating an entity reference

The following VB code creates a CDATA section in an element Node in the DOMDocument:

```
objXMLDOMEntityReference = objElement.createEntityReference(name)
```

This creates an entity reference with a specified case-sensitive name. This method is important when you want to create a Node that needs to have the entity reference in the XML. The entity is defined in the DTD. In our DTD example (people2.dtd), we have specified the following entity:

```
<!ENTITY UK "United Kingdom">
```

In the following example, we need to create an address element for a person. However, our example is a bit more complicated, because the entity UK needs to come at the end of the address. Therefore, we need to create an element for the address, plus we need to create an entity reference Node for the address element. This is how we need to combine the two:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objPerson As IXMLDOMNode
```

```
Dim objChild As IXMLDOMNode
```

```
Dim objEntityRef As IXMLDOMEntityReference
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.resolveExternals = True
```

```
objDOMDocument.validateOnParse = True
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
Set objPerson = objDOMDocument.createElement("PERSON")
```

```
objDOMDocument.documentElement.appendChild objPerson
```

```
Set objChild = objDOMDocument.createElement("ADDRESS")
```

```
Set objEntityRef = objDOMDocument.createEntityReference("UK")
```

```
objChild.Text = "34 Erica Street, Isle of Dogs,"
```

```
objChild.appendChild objEntityRef
```

```
objPerson.appendChild objChild
```

Ø Load a DOMDocument from file.

Ø Create a person element and add it to the documentElement NodeList.

- ∅ Create the child Node for the Address element.
- ∅ Create an entity reference Node, which we know is the “UK” entity from our DTD.
- ∅ To our address element (objChild), add the address text, excluding the entity reference.
- ∅ Append the entity reference object to the address element as a child Node. This creates the correct syntax for an element that has normal text, plus an entity reference in it.

When running this example, after the line

```
objChild.appendChild objEntityRef
```

the objChild xml and text properties are as follows:

```
objChild.xml: <ADDRESS>34 Erica Street, Isle of Dogs, &UK;</ADDRESS>
```

```
objChild.text: 34 Erica Street, Isle of Dogs, United Kingdom
```

createNode()

Is a member of: DOMDocument

Syntax

As an alternative to the createElement(), createAttribute(), etc. methods, you can also use the createNode() method, which creates any type of Node, using the following syntax:

```
set objXMLDOMNode = objDOMDocument.createNode(Type, name, namespaceURI)
```

Table describes the syntax of createNode().

Signature of the createNode method (continued)

Type	Specify the type of Node, which can be the numeric value of the nodeType or a string of the nodeTypeString.
name	Specify the name of the Node or, if there is no name, then an empty string (“”). Remember that if you’re dealing with an element or an entity reference, the name is case-sensitive.
namespaceURI	Specify the namespace or, if there is no namespaceURI, then an empty string (“”).

Remark

Before we continue, please refer to the `nodeType` property section, which explains in detail the different Node types, their enumeration values, and descriptions.

You cannot create a Node of the following types:

Ø Document—it is the root of the XML; it cannot be a childNode. The `createNode()` method only creates child Nodes. To create the root of your XML, you initialize a `DOMDocument` object as we have seen in most examples so far.

Ø Document type—DTDs cannot be added to the `DOMDocument`. See the `doctype` property in this section.

Ø Entity—entities are declared in the DTD; therefore, being external to the XML file restricts their creation

Ø Notation—this is the same as entities

If you attempt to create any of the above, an error will occur.

Table lists what to insert in the Type and Name parts of the `createNode()` signature. We have not added the namespace property in the signature, as this only applies to element and attribute Nodes, where you can specify the namespace. If there is not a namespace, you need to add an empty string (“”), as this signature has no optional addresses.

In the Type part of the `createNode()` signature, you can insert either the `nodeType` value or the `nodeTypeString` value. You can insert the Name section in the name part of the signature.

Specifications of values to insert in `createNode` signature (continued)

Type of Node	node Type value	nodeType String value	name
Element	1	“element”	tagName/nodeName (e.g., “PERSON”)
Attribute	2	“attribute”	name of the attribute/nodeName (e.g., “PERSONID”)
Text	3	“text”	empty string (“”)
CDATA section	4	“cdatasection”	empty string (“”)
Entity Reference	5	“entityreference”	name of the referenced entity/nodeName (e.g., “USA”)
	7	“processinginstruction”	target/nodeName (“realaudio”)

Processing Instruction			
Comment	8	"comment"	empty string ("")
Document Fragment	11	"documentfragment"	empty string ("")

Example

In the following examples, we create the different nodeTypes and append them to the documentElement property of the DOMDocument.

To create an element Node, pass the nodeTypeString as the type:

```
Set objNode = m_objXMLDOM.createElement("element", "PERSON", "")
```

```
m_objXMLDOM.documentElement.appendChild objNode
```

Ø It is such a pity that the DOM- NodeType enum for the Node type in the signature was not used as the passed variable. (Instead, an integer or string was inserted.) It would have made the Intellisense auto-complete in VB so much easier to read.

Alternatively, you can pass the NodeType enumeration as the type:

```
Set objNode = m_objXMLDOM.createElement(NODE_ELEMENT, "PERSON", "")
```

```
m_objXMLDOM.documentElement.appendChild objNode
```

Otherwise, you can pass the NodeType enumeration value as the type:

```
Set objNode = m_objXMLDOM.createElement(1, "PERSON", "")
```

```
m_objXMLDOM.documentElement.appendChild objNode
```

The example from the createComment() method example would change as follows for the createNode() method:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objComment As IXMLDOMComment
```

```
Set objDOMDocument = New DOMDocument
```

```
Set objComment = objDOMDocument.createNode(NODE_COMMENT, "", "")
```

```
objComment.Text = "This is a comment."
```

`objDOMDocument.appendChild objComment`

Ø We've used the `nodeType` enumeration for the `nodeType`. Don't forget to pass empty strings for the other two parameters, as they are not optional.

Ø Add the comment text.

Ø Append the comment Node to its parent, which in this case is the `DOMDocument`.

getAttribute()

Is a member of: `XMLDOMElement`

Syntax

```
vntValue = objXMLDOMElement.getAttribute(strAttributeName)
```

Remark

If your element has an associated attribute, you can look up the string value of that attribute its name. It then returns the attributes value (`nodeValue`) in a vari-ant datatype.

Example

In the following example, we have already loaded our `people2.xml` file into a `DOMDocument` (`objDOMDocument`). We add a new `Person` element and its attribute. Once we have added this element, we get the value of the attribute for this element.

```
Dim objPerson As IXMLDOMElement
```

```
Dim objAttrib As IXMLDOMAttribute
```

```
Dim vntValue As Variant
```

```
Set objPerson = objDOMDocument.createElement("PERSON")
```

```
Set objAttrib = objDOMDocument.createAttribute("PERSONID")
```

```
objAttrib.Text = "p7"
```

```
objPerson.Attributes.SetNamedItem objAttrib
```

```
objDOMDocument.documentElement.appendChild objPerson
```

```
vntValue = objPerson.getAttribute("PERSONID")
```

`vntValue` returns "p7," the value of the attribute.

Ø As an alternative to this method, see the `nodeFromID()` method for getting an attribute when you know the id value but you want to find its associated Node.

getAttributeNode()

Is a member of: XMLDOMElement

Syntax

```
objXMLDOMAttribute = objXMLDOMElement.getAttributeNode(strAttributeName)
```

Remark

This method is the same as the `getAttribute()` method, except that it returns an attribute object instead of the attribute's value. If your element has an associated attribute, you look up the attribute Node object using the name of the attribute (`nodeName`). If NULL is returned, then there is no attribute for the Node.

Example

Using the same example as in `getAttribute()` method, we change the last line so that it returns an attribute object.

```
Dim objPerson As IXMLDOMElement  
  
Dim objAttrib As IXMLDOMAttribute  
  
Dim strValue As String  
  
Set objPerson = objDOMDocument.createElement("PERSON")  
  
Set objAttrib = objDOMDocument.createAttribute("PERSONID")  
  
objAttrib.Text = "p7"  
  
objPerson.Attributes.SetNamedItem objAttrib  
  
objDOMDocument.documentElement.appendChild objPerson  
  
Set objAttrib = objPerson.getAttribute("PERSONID")  
  
Ø objAttrib returns the attribute in the objPerson element.
```

getElementsByTagName()

Is a member of: DOMDocument

XMLDOMElement

Syntax

```
set objXMLDOMNodeList = objDOMDocument.getElementsByTagName(tagname)
```

Remark

The parameter Tagname in the `getElementsByTagName()` signature is a string, specifying the element name to find. If you specify that the tagname is an asterisk ("*"), then all the elements are returned in the `DOMDocument`.

The returned `NodeList` of this method is different from the `nodeLists` that we have dealt with so far. You normally work with a `NodeList` that returns a collection from the `documentElement`, such as:

```
set objXMLDOMNodeList = objDOMDocument.documentElement
```

This will return a `NodeList` collection that returns its child Nodes as shown in figure .

You can call `getElementsByTagName()` from the `DOMDocument` root, specifying that you want everything (*):

```
set objXMLDOMNodeList = objDOMDocument.getElementsByTagName("*")
```

In the above example, `getElementsByTagName()` returns a `NodeList` collection that has child Nodes as follows in figure :

This method returns a `NodeList` object in which each child Node is grouped by their element names, instead of being grouped by the normal `TreeView` effect.

Example

To get a clearer view of what we are trying to explain, look in your Local Views in VB for the following examples. See the difference in the collections of the `NodeList` object by calling:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNodeList As IXMLDOMNodeList
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objNodeList = objDOMDocument.documentElement
```

Ø Now look at the returned NodeList object in the Local Views in VB. Try the following:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNodeList As IXMLDOMNodeList
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objNodeList = objDOMDocument.getElementsByTagName("*")
```

Ø You can use this method when you want to get a specific Node, like NAME. However, take note that it returns a NodeList object; therefore, you need to iterate through the returned Nodes.

An example of this is when we want to put the NAME values of our element into a text box.

```
Set objNodeItems = objNode.getElementsByTagName("NAME")
```

```
txtName.Text = objNodeItems.Item(0).nodeValue
```

Ø We indicate that we want the "NAME" element returned from the current Node.

Ø We want to get the value from the first Node in our collection.

For an easier method, look at the selectSingleNode() method later in this section.

getNamedItem()

Is a member of: XMLDOMNamedNodeMap

Syntax

```
Set objXMLDOMNode = objXMLDOMNamedNodeMap.getNamedItem(strAttributeName)
```

Remark

The XMLDOMNamedNodeMap object is used to find and manipulate attributes for a Node, although the XMLDOMElement interface gives you many methods as well to do this.

This method takes the name of an attribute to find the Node associated with that attribute. However, to be able to do this, we first need to get a hold of the XMLDOMNamedNodeMap object, which is the attributes property of a Node. If there are attributes in the current Node, then we can query the XMLDOMNamedNodeMap collection.

Example

In the following example, we want to store the value of the elements ID attribute in the tag of a TreeView. We have already loaded a DOMDocument from our people2.xml. We have passed a person element in the following example.

```
Dim objNode As IXMLDOMNode
```

```
Dim objAttributes As IXMLDOMNamedNodeMap
```

```
Dim objAttributeNode As IXMLDOMNode
```

```
Set objAttributes = objNode.Attributes
```

```
If objAttributes.length > 0 Then
```

```
    Set objAttributeNode = objAttributes.getNamedItem("PERSONID")
```

```
    tvwElement.Tag = objAttributeNode.nodeValue
```

```
End If
```

Ø Get the XMLDOMNamedNodeMap from the Nodes attribute property.

Ø Check that there are available attributes.

Ø Our id reference is "PERSONID"; therefore, tell the NameNodeListMap to get this Node by using the getNamedItem method.

hasChildNodes()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMEElement

XMLDOMEntity
XMLDOMEntityReference
XMLDOMNotation
XMLDOMProcessingInstruction
XMLDOMText
XTLRuntime

Syntax

```
blnValue = objXMLDOMNode.hasChildNodes()
```

Remark

This method is read-only. This method returns a boolean indicating whether the current Node has any children. It can be used instead of the length property, as explained earlier in this chapter.

Example

We have snatched the following example from the length property and changed it to show how to use the hasChildNodes() method.

In this example, we test the hasChildNodes() method to see if a DOMDocument has loaded properly. If the hasChildNodes() returned True, then we can proceed with the rest of the code working with the DOMDocument object.

```
Dim objDOMDocument As DOMDocument  
  
Dim objNode As IXMLDOMNode  
  
Set objDOMDocument = New DOMDocument  
  
objDOMDocument.async = False  
  
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"  
  
If objDOMDocument.hasChildNodes Then  
    'run rest of code here  
  
End If
```

Our code has loaded the XML from our main XML example. The `hasChildNodes()` method returns `True`; therefore, we can proceed with our `DOMDocument`.

insertBefore()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

```
set objXMLDOMNode = objXMLDOMNode.insertBefore(newChild, refChild)
```

Remark

This method inserts a new child Node before the current Node (`objXMLDOMNode`).

From the methods signature, if you do not include a parameter, `refChild`, then this function acts the same as the `appendChild()` method. Here the `ref-Child` must be a child Node of the current Node (`objXMLDOMNode`). This indicates that the new child must be inserted before this `refChild` Node (which is the left sibling).

The DOMDocument checks that the refChild is a child Node of the current Node (objXMLDOMNode). If it is not, then an error will occur.

This method returns the child that has been successfully inserted. The method will return an error if you have tried to do an impossible operation, such as adding a Node of type NODE_DOCUMENT_TYPE (which cannot be a child Node), or the ref-Child is not a child Node of the current Node. See the nodeType property and the createNode() method for more information on types of Nodes.

load()

Is a member of: DOMDocument

Syntax

```
blnValue = objDOMDocument.load(url)
```

Remark

When you load a DOMDocument, any existing contents in the DOMDocument are discarded.

This is one of the two common ways to load XML into a DOMDocument. In the methods signature, the URL is a string that is the location of an XML file. True or False is returned to indicate success or failure of the load.

See the async property for related information on using this method and an example.

loadXML()

Is a member of: DOMDocument

Syntax

```
blnValue = objDOMDocument.loadXML(xmlString)
```

Remark

When you call the loadXML() method for a DOM-Document, any existing contents in the DOMDocument are discarded.

This is one of the two common ways to load XML into a DOMDocument (see the load() method). In the loadXML signature, xmlString is either a string that is an entire XML document or a well-formed fragment.

True or False is returned to indicate success or failure of the load.

loadXML bug in XML 2.0

In the Microsoft XML 2.0 object, there is a bug in the loadXML() method. Unfortunately, when used from VB or VBScript, your code may return an error that says that there is a parse error. The error usually complains that you are using an element in your XML that is not declared in the DTD you have referenced.

The error looks something like this:

The element “PEOPLE” is used but not declared in the DTD/Schema.

The reason is that the loadXML() method, when called from VB or VBA, will always act as if the resolveExternals property of the document was false. There is no work-around, but it will be fixed for the next release of the Microsoft XML objects.

nextNode()

Is a member of: XMLDOMNodeList

XMLDOMNamedNodeMap

Syntax

```
set objXMLDOMNode = objXMLDOMNodeList.nextNode()
```

Remark

This method is used to iterate through either the collection returned from child Node of an element (XMLDOMNodeList collection) or the collection returned from the attributes property of an element (XMLDOMNamedNodeMap collection).

The pointer after returning either of these collections is set to before the first Node in the list. Therefore, when you call the nextNode() method for the first time, it returns the first Node in the list.

As you iterate through this collection, NULL will be returned when the current Node is the last Node or there are no items in the list.

Example

In previous examples, we showed you a similar example of iterating through the Nodes in an XMLDOMNodeList collection:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNode As IXMLDOMNode
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
For Each objXMLDOMNode In objDOMDocument.childNodes
```

```
    'do something with this
```

```
Next
```

Ø Let's change this example to demonstrate using the nextNode property:

```
Dim objDOMDocument As DOMDocument
```

```
Dim objNodeList As IXMLDOMNodeList
```

```
Dim objXMLDOMNode As IXMLDOMNode
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
Set objNodeList = objDOMDocument.childNodes
```

```
Set objXMLDOMNode = objNodeList.nextNode
```

```
Do Until objXMLDOMNode Is Nothing
```

```
    ' do something with this node
```

```
    Set objXMLDOMNode = objNodeList.nextNode4
```

```
Loop
```

Ø Get a reference to the XMLDOMNodeList collection.

Ø Fetch the first Node.

Ø Prepare your loop, exiting once the last Node has been fetched.

Ø Fetch the next Node.

nodeFromID()

Is a member of: XMLDOMDocument

Syntax

```
set objXMLDOMNode = objDOMDocument.nodeFromID(strId)
```

Remark

The nodeFromID() method was designed to work specifically with ID and IDREF type attributes. (See the attributes property in this section.) Because this method is only available for the XMLDOMDocument interface, it will find any child Node that has the ID value that you have specified. If no child Node is found, it returns NULL.

Example

In the following example, we have stored the PERSONID attribute in the TreeView tag. When the user clicks on the TreeView, we fetch the Node in the DOMDocument using this method.

```
Dim objPersonElement As IXMLDOMElement
```

```
If Trim(objSelNode.Tag) <> "" Then
```

```
    Set objPersonElement = m_objDOMPeople.nodeFromID(objSelNode.Tag)
```

```
    lblElement.Caption = objPersonElement.nodeName & ": " & _
```

```
objPersonElement.Attributes(0).nodeValue
```

```
End If
```

Ø Check that this is not a TreeView Node that has an empty tag.

Ø Find the Node in the DOMDocument that has the tag value.

Ø Set a label's caption with details from the found Node.

open()

Is a member of: XMLHTTPRequest

Syntax

```
objHTTPRequest.open
```

Remark

The `open()` method is used extensively in the examples in this book. It is used to open a connection to a web server so that you can use the `send()` method to use a `GET()`, `PUT()`, or similar HTTP command.

Example

```
Dim objXMLHttp As New XMLHttpRequest
```

```
Dim objDOMDocument As DOMDocument
```

```
objXMLHttp.Open "POST", "http://localhost/xmlcode/demo.asp", False
```

```
objXMLHttp.Send
```

```
Set objDOMDocument= objXMLHttp.responseXML
```

Ø Open the POST (or GET) connection to the web server.

Ø Establish the connection.

Ø Receive the response—note there are different types of responses; binary and text are also supported.

removeAttribute()

Is a member of: XMLDOMElement

Syntax

```
objXMLDOMElement.removeAttribute strAttributeName
```

Remark

If your element has an associated attribute, you can remove it from the Node by specifying the attribute's name. This method does not return anything. If you try to remove an attribute that does not exist, nothing happens and there is no warning.

Example

In the following example, we remove the `PERSONID` attribute from the first child Node of the `documentElement` child Node collection.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMElement As IXMLDOMElement
```

```
Dim strFirstValue As String
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMEElement = objDOMDocument.documentElement.firstChild
```

```
objXMLDOMEElement.removeAttribute "PERSONID"
```

removeAttributeNode()

Is a member of: XMLDOMEElement

Syntax

```
set objXMLDOMAttribute = objXMLDOMEElement.removeAttributeNode(objXMLDOMAttribute)
```

Remark

You can remove an attribute from a Node if you have an associated XMLDOMAttribute Node type for the element Node. If the attribute was successfully removed, it will return the removed attribute Node. If not, it will return NULL.

Example

Using the example from the removeAttribute() method, we will show how to do the same function using the removeAttributeNode() method. Here we need to first get a reference to the attribute before we can remove it. The difference with this method is that we can test if the removal of the attribute was successful.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMEElement As IXMLDOMEElement
```

```
Dim objAttrib As IXMLDOMAttribute
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMEElement = objDOMDocument.documentElement.firstChild
```

```
Set objAttrib = objXMLDOMEElement.removeAttributeNode(objXMLDOMEElement.getAttributeNode("PERSONID"))
```

If objAttrib Is Nothing Then

' this removal was unsuccessful, show warning

End If

Ø We have chosen to use the `getAttributeNode()` method to get a reference to the attribute that we want to remove. The returned attribute (`objAttribute`) from the method is the removed attribute.

Ø If the attribute removal was successful, then show a warning.

removeChild()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

Set objXMLDOMNode = objXMLDOMNode.removeChild(oldChild)

Remark

The parameter `oldChild` is the Node to be removed from the list of children for this Node. If the removal is successful, this method returns the removed child Node (`objXMLDOMNode`).

Although this method has been added to most of the `DOMDocument` inter-faces, it cannot actually be used with all of them, as they may not have child Nodes to remove.

This method returns the Node that has just been removed (see the `nodeType` property for more information on Node types). It works successfully with the following Node types:

- Ø Document
- Ø Element
- Ø Attribute
- Ø Document Fragment
- Ø Node (if its type is one of the above)

The following types of Nodes cannot have `childNodes`; therefore, this method cannot be used successfully for the following `nodeTypes`:

- Ø Comment
- Ø Text
- Ø Processing Instruction
- Ø CDATA section
- Ø Notation

Although the following can have children, they are `childNodes` that cannot be removed, as they may be part of a DTD:

- Ø Document type
- Ø Entity
- Ø Entity reference

Example

The following example removes the firstChild of the child Node collection of the documentElement property.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMEElement As IXMLDOMEElement
```

```
Dim objRemovedElement As IXMLDOMEElement
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
```

```
Set objXMLDOMEElement = objDOMDocument.documentElement.firstChild
```

```
Set objRemovedElement = objDOMDocument.documentElement.removeChild(objXML-DOMEElement)
```

```
If objRemovedElement Is Nothing Then
```

```
    ' this removal was unsuccessful, show warning
```

```
End If
```

Ø The object objRemovedElement returns the removed Node if the removeChild execution was successful.

Ø If the removal was unsuccessful, then it returns a warning.

removeNamedItem()

Is a member of: XMLDOMNamedNodeMap

Syntax

```
set objXMLDOMNode = objXMLDOMNamedNodeMap.removeNamedItem(strAttribute)
```

Remark

Here we repeat our introduction to getNamedItem(). The XMLDOMNamedNodeMap object is used to find and manipulate attributes for a Node, although the XML-DOMEElement interface gives you many methods as well to do this.

For all the remove type meth-ods, the Node may be removed from the instantiatedDOMDocument—but it is not yet removed from the actual XML file—until you have called the save() method

This method takes the name of an attribute to remove the attribute from an element Node. It is similar to the `removeAttribute()` method, except that it returns the removed attribute, and it's a method for a different object.

Example

In the following example, we get a reference to an `XMLDOMNamedNodeMap` object, in order to use the `removeNamedItem()` method.

```
Dim objDOMDocument As DOMDocument

Dim objNamedNodeMap As IXMLDOMNamedNodeMap

Dim objAttrib As IXMLDOMAttribute

Set objDOMDocument = New DOMDocument

objDOMDocument.async = False

objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"

Set objNamedNodeMap = objDOMDocument.documentElement.firstChild.Attributes

Set objAttrib = objNamedNodeMap.removeNamedItem("PERSONID")

If objAttrib Is Nothing Then

    ' this removal was unsuccessful, show warning

End If
```

Ø This method can also be done in one line of code:

```
Ø Set objAttrib = objDOMDocument.documentElement.firstChild.Attributes.removeNamedItem("PERSONID")
```

Ø Get a reference to the `XMLDOMNamedNodeMap` object, which is returned from the `attributes` property.

Ø Remove the attribute using the `attributes` name.

replaceChild()

Is a member of: DOMDocument

XMLDOMNode
XMLDOMAttribute
XMLDOMCDATASection
XMLDOMComment
XMLDOMDocumentFragment
XMLDOMDocumentType
XMLDOMElement
XMLDOMEntity
XMLDOMEntityReference
XMLDOMNotation
XMLDOMProcessingInstruction
XMLDOMText
XMLRuntime

Syntax

```
set objXMLDOMNode = objXMLDOMNode.replaceChild(newChild, oldChild)
```

Remark

This method replaces one child Node with another child Node, instead of removing the old child and then appending the new child.

The parameter newChild is the Node that will replace the oldChild Node. If newChild is NULL, oldChild is removed without a replacement. The oldChild that had been replaced is returned in the objXMLDOMNode object, if the execution of this method is successful.

Again, take note of the types of Nodes you are dealing with. Some types of Nodes cannot be children, and therefore cannot be used with this method.

reset()

Is a member of: XMLDOMNodeList
XMLDOMNamedNodeMap

Syntax

objXMLDOMNodeList.reset()

Remark

This method resets the iterator to point before the first Node in the XMLDOMNodeList object, so that the next call to nextNode() returns the first item in the list.

save()

Is a member of: XMLDOMDocument

Syntax

objDOMDocument.save(objTarget)

Remark

The parameter objTarget can be a file name, an ASP response object, an XML document object, or a custom object that supports persistence and specific interfaces.

If you have manipulated the DOMDocument in any way by appending, removing, or whatever, remember that it is not saved to the XML file until you have called the save() method.

The save() method does not return anything; however, if an error is reported, the error can expose one of the return values listed in table .

save() method return values (continued)

Returned	Description
S_OK	Success
XML_BAD_ENCODING	The document contains a character that does not belong in the specified encoding
E_INVALIDARG	A string was provided but it is not a valid file name. or an object was provided that does not support any of the above interfaces
E_ACCESSDENIED	save() operation is not permitted
E_OUTOFMEMORY	save() cannot allocate buffers
Other values	Any other file system error

Saving in ADO 2.5

You may have noticed in previous examples that we tend to save our XML to a file and then load the file into a DOMDocument. In Microsoft ADO 2.5 that ships with Windows 2000, you will be able to save directly to a DOMDocument.

This is how you will be able to do it:

```
Dim strSQL As String

Dim adoCon As ADODB.Connection

Dim adoRst As ADODB.Recordset

Dim objDOMDocument As DOMDocument

strSQL = "SELECT * FROM categories"

Set adoCon = New ADODB.Connection

Set adoRst = New ADODB.Recordset

adoCon.ConnectionString = "northwind"

adoCon.CursorLocation = adUseClient

adoCon.Open

Set adoRst.ActiveConnection = adoCon

adoRst.Open strSQL, , adOpenForwardOnly, _
adLockReadOnly, adCmdText

Set objDOMDocument = New DOMDocument

objDOMDocument.async = False

adoRst.Save objDOMDocument, 1 ' save this to DOM

adoRst.Close

adoCon.Close

Set adoCon = Nothing
```

- Ø Prepare ADO objects.
- Ø Open database connection.
- Ø Get resultset.
- Ø Prepare the DOMDocument.
- Ø Save the recordset to the DOMDocument.

selectNodes()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

Set objXMLDOMNode = objDOMDocument.selectNodes(strXQLQuery)

Remark

For more information on XSL pattern matching, see chapter 5, “XSL—adding style to XML.”

This method returns an `XMLDOMNodeList` collection of all the Nodes for a specified XQL query.

Example

In the following example, we retrieve an `XMLDOMNodeList` collection of all the `NAME` elements from our `DOMDocument` that are in the `PERSON` element.

```
Dim objDOMDocument As DOMDocument
Dim objNodeList As IXMLDOMNodeList
Set objDOMDocument = New DOMDocument
objDOMDocument.async = False
objDOMDocument.Load "http://localhost/xmlcode/people2.dtd"
Set objNodeList = objDOMDocument.selectNodes("//PERSON//NAME")
```

selectSingleNode()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
Set objXMLDOMNode = objXMLDOMELEMENT.selectSingleNode(strXQLQuery)
```

Remark

Once again, as in the `selectNodes()` method, `selectSingleNode()` uses XQL to find the required Node. However, it only returns a single Node, `XMLDOMNode` object, which is the first Node that matches the pattern if it's found. If no match is found, it returns `NULL`.

Example

The `selectNodes()` and `selectSingleNode()` methods can also be used to search for Nodes that have certain attribute values. As we mentioned, you will need to do a bit of research to work out how to work with XQL, which we do not cover in detail here.

In the following example, we iterate through all the `childNodes` in the `documentElement` collection. We only want to find the `NAME` element in each `PERSON` element.

```
Dim objDOMDocument As DOMDocument
```

```
Dim objXMLDOMNode As IXMLDOMNode
```

```
Dim objElement As IXMLDOMELEMENT
```

```
Set objDOMDocument = New DOMDocument
```

```
objDOMDocument.async = False
```

```
objDOMDocument.Load "http://localhost/xmlcode/people2.xml"
```

```
For Each objXMLDOMNode In objDOMDocument.documentElement.childNodes
```

```
    Set objElement = objXMLDOMNode.selectSingleNode("NAME")
```

```
    ' do something with this name element
```

```
Next
```

Look at `getElementsByTagName()`, which does almost the same thing as this method. The beauty of this method, however, is that it returns a single Node instead of an `XMLDOMNodeList`. Therefore, this method is

easier to work with, as you don't need to iterate through a NodeList collection to just get the one value out of it.

send()

Is a member of: XMLHttpRequest

Syntax

```
objHttpRequest.send([varBody])
```

Remark

The send() method is used extensively in the examples in this book. Once you have opened a connection to a web server with the open method, you can use the send method to communicate with the webserver and then use PUT(), GET(), or similar HTTP commands against the webserver.

Example

```
Dim objXMLHttp As new XMLHttpRequest
```

```
Dim objDOMDocument As DOMDocument
```

```
objXMLHttp.Open "POST", "http://localhost/xmlcode/demo.asp", False
```

```
objXMLHttp.Send
```

```
Set objDOMDocument= objXMLHttp.responseXML
```

Ø Open the POST (or GET) connection to the web server.

Ø Establish the connection.

Ø Receive the response—note there are different types of responses; binary and text are also supported.

setAttribute()

Is a member of: XMLDOMElement

Syntax

```
objXMLDOMElement.setAttribute strAttributeName, vntValue
```

Remark

There are so many ways to add an attribute to a DOMDocument. Here are a few:

- Ø setAttribute
- Ø setAttributeNode
- Ø setNamedItem
- Ø appendChild

We find this method, `setAttribute()`, the easiest way to add an attribute. However, you need to have a reference to the element interface before you can add the attribute. You can directly give the attribute its most important values, the name of the attribute (`strAttributeName`) and its value (`vntValue`), in one call through this method.

Example

In the following example, we add another PERSON element to our document— Element in a `DOMDocument`, which needs to have an attribute.

```
Dim objPerson As IXMLDOMElement  
  
Set objPerson = m_objDOMPeople.createElement("PERSON")  
  
objPerson.setAttribute "PERSONID", "p7"  
  
m_objDOMPeople.documentElement.appendChild objPerson
```

- Ø Create the PERSON element.
- Ø Add the attribute to this element.
- Ø Add this new element with its attribute to the `DOMDocument`.

setAttributeNode()

Is a member of: XMLDOMElement

Syntax

```
set objXMLDOMAttribute = objXMLDOMElement.setAttributeNode(objXMLDOMAttribute)
```

Remark

As we mentioned, this is one of the methods used to add or manipulate an attribute. As with the `setAttribute()` method, you can only use this method with the `XMLDOMElement` interface. However, instead of passing a string, you need to pass an `XMLDOMAttribute` object when adding the attribute to the element. This attribute Node is added to the element after using the `createAttribute()` method to create the attribute.

Example

By modifying the example for the `setAttribute()` method, we will show how the `setAttributeNode()` method differs from the `setAttribute()` method:

```
Dim objPerson As IXMLDOMElement
Dim objAttrib As IXMLDOMAttribute
Set objPerson = m_objDOMPeople.createElement("PERSON")
Set objAttrib = m_objDOMPeople.createAttribute("PERSONID")
objAttrib.text = "p7"
objPerson.setAttributeNode objAttrib
m_objDOMPeople.documentElement.appendChild objPerson
```

- Ø Create the PERSON element.
- Ø Create the PERSONID element.
- Ø Give the attribute Node its value.
- Ø Add the attribute to the PERSON element.
- Ø Add this new element with its attribute to the DOMDocument.

setNamedItem()

Is a member of: XMLDOMNamedNodeMap

Syntax

```
set objXMLDOMNode = objXMLDOMNamedNodeMap.setNamedItem(objXMLDOMNode)
```

Remark

This is true for all the methods used to add/manipulate an attribute: if you add an attribute with the same name as an existing attribute, it will replace the existing attribute.

Surprise, this is yet another method to add/manipulate an attribute to the DOM-Document. This time it is done using the `XMLDOMNamedNodeMap` interface.

You don't need to create an `XMLDOMNamedNodeMap` object to use this method, because the `attributes` property of a `Node` is a `XMLDOMNamedNodeMap` object.

Example

In the following example, we show how to add an attribute using this method.

```
Set objPerson = m_objDOMPeople.createElement("PERSON")
```

```
Set objAttrib = m_objDOMPeople.createAttribute("PERSONID")
```

```
objAttrib.text = "p7"
```

```
objPerson.attributes.setNamedItem objAttrib
```

```
m_objDOMPeople.documentElement.appendChild objPerson
```

Ø Create an attribute Node.

Ø Add the attribute's key details.

Ø The attributes property returns a NodeList object; therefore you can use the setNamedItem() method.

Ø Add this new "PERSON" element to the DOMDocument.

transformNode()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMEElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation

XMLDOMProcessingInstruction

XMLDOMText

XTLRuntime

Syntax

```
strValue = objDOMDocument.transformNode(strStylesheet)
```

Remark

The transformNode() method applies a stylesheet to the current Node and its child Nodes. It returns this data in a string, which is very useful for when you want to pass your data back as HTML (which means that your XSL file needs to convert your XML data to HTML):

```
strValue = objDOMDocument.transformNode(strStylesheet)
```

The string strValue will contain the contents to transform the Node into the required string. For more information, see chapter 5, “XSL—adding style to XML.”

Example

The following example shows the string from the transformed Node being passed back using the transformNode() method. We use people2.xml to demonstrate the transformNode() method.

Our XSL example looks as follows:

```
<?xml version="1.0"?>  
  
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">  
  
<xsl:template match="/">  
  
<HTML>  
  
<BODY>  
  
<TABLE BORDER="2">  
  
<TR>  
  
<TD>Name</TD>  
  
<TD>Address</TD>  
  
<TD>Tel</TD>
```

```
<TD>Fax</TD>

<TD>Email</TD>

</TR>

<xsl:for-each select="PEOPLE/PERSON">

<TR>

  <TD><xsl:value-of select="NAME"/></TD>

<TD><xsl:value-of select="ADDRESS"/></TD>

<TD><xsl:value-of select="TEL"/></TD>

<TD><xsl:value-of select="FAX"/></TD>

<TD><xsl:value-of select="EMAIL"/></TD>

</TR>

</xsl:for-each>

</TABLE>

</BODY>

</HTML>

</xsl:template>

</xsl:stylesheet>
```

The code is:

```
Dim objXMLStyle As New XmlDocument
```

```
Dim objDOMDocument As New XmlDocument
```

```
Dim strXMLFromXSL As String
```

```
Call objDOMDocument.Load("http://localhost/xmlcode/people2.xml")
```

```
Call objXMLStyle.Load("http://localhost/xmlcode/people.xsl")
```

```
strXMLFromXSL = objDOMDocument.transformNode(objXML-Style.documentElement)
```

The strXMLFromXSL string returns the following HTML:

```
<HTML>
<BODY>
<TABLE BORDER="2">
<TR>
<TD>Name</TD>
<TD>Address</TD>
<TD>Tel</TD>
<TD>Fax</TD>
<TD>Email</TD>
</TR>
<TR>
<TD>Mark Wilson</TD>
<TD>911 Somewhere Circle, Canberra, Australia</TD>
<TD>(++612) 12345</TD>
<TD>(++612) 12345</TD>
<TD>markwilson@somewhere.com</TD>
</TR>
<TR>
<TD>Tracey Wilson</TD>
<TD>121 Zootle Road, Cape Town, South Africa</TD>
<TD>(++2721) 531 9090</TD>
```

<TD>(+2721) 531 9090</TD>

<TD>Tracey Wilson@somewhere.com</TD>

</TR>

<TR>

<TD>Jodie Foster</TD>

<TD>30 Animal Road, New York, USA</TD>

<TD>(+1) 3000 12345</TD>

<TD>(+1) 3000 12345</TD>

<TD>Jodie Foster@somewhere.com</TD>

</TR>

<TR>

<TD>Lorin Maughan</TD>

<TD>1143 Winners Lane, London, United Kingdom</TD>

<TD>(+94) 17 12345</TD>

<TD>+94) 17 12345</TD>

<TD>Lorin Maughan@somewhere.com</TD>

</TR>

<TR>

<TD>Steve Rachel</TD>

<TD>90210 Beverly Hills, California, United States of America</TD>

<TD>(+1) 2000 12345</TD>

<TD>(+1) 2000 12345</TD>

<TD>Steve Rachel@somewhere.com</TD>

</TR>

</TABLE>

</BODY>

</HTML>

transformNodeToObject()

Is a member of:

- DOMDocument
- XMLDOMNode
- XMLDOMAttribute
- XMLDOMCDATASection
- XMLDOMComment
- XMLDOMDocumentFragment
- XMLDOMDocumentType
- XMLDOMElement
- XMLDOMEntity
- XMLDOMEntityReference
- XMLDOMNotation
- XMLDOMProcessingInstruction
- XMLDOMText
- XTLRuntime

Syntax

```
objDOMDocument.transformNode(objXMLStyle.documentElement, vntNewDOMDocument)
```

Remark

This is similar to the transformNode() method; however, it doesn't actually change the underlying DOMDocument. Instead it returns a variant (vntNewDOM-Document) with a new transformed DOMDocument that has the stylesheet applied to it.

XPath Reference

General intro

An XPath expression contains one or more "location steps", separated by slashes. Each location step has the following form:

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

Or in English: an axis name, then two colons, then a node-test and finally zero or more predicates in brackets. We will show a list of valid axes and a list of valid node-tests in this appendix. A predicate is an expression. It exists of values, operators and other XPath expressions.

The XPath axis contains a part of the document, defined from the perspective of the "context node". The node test makes a selection from the nodes on that axis. By adding predicates, it is possible to select a subset from these nodes. If the expression in the predicate returns true, the node remains in the selected set, otherwise it is removed.

XPath defines a set of functions for use in predicates. These are listed in the appendix as well.

Axes

ancestor

Description: Contains the context node's parent node, its parent's parent node etc., all the way up to the document root. If the context node is the root node, the ancestor node is empty.

Primary node type: element

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

ancestor-or-self

Description: Identical to the ancestor axis, but including the context node itself.

Primary node type: element

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

attribute

Description: Contains all attributes on the context node. The axis will be empty unless the context node is an element.

Primary node type: attribute

Shorthand: @
Implemented [w3c1](#), [msxml26](#) (erroneously returns namespace declarations as well) , [msxml3](#), [msxml4](#), [msxml.NET](#)

child

Description: Contains all direct children of the context node (that is the children, but not any of the children's children)
Primary node type: element
Shorthand: (default axis)
Implemented [w3c1](#), [msxml2](#) (only shorthand syntax) , [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

descendant

Description: All children of the context node, including all children's children recursively
Primary node type: element
Shorthand: //
Implemented [w3c1](#), [msxml2](#) (only shorthand syntax) , [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

descendant-or-self

Description: Identical to the descendant axis, but including the context node itself.
Primary node type: element
Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

following

Description: Contains all nodes that come after the context node in the document order. This means that the start tag of the node must come after the closing tag of the context node and therefore excludes the descendants of the context node.
Primary node type: element
Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

following-sibling

Description:	Contains all siblings (children of the same parent node) of the context node that come after the context node in document order
Primary node type:	element
Implemented	w3c1 , msxml3 , msxml4 , msxml.NET

namespace

Description:	Contains all namespaces available on the context node. This includes the default namespace and the xml namespace (these are automatically declared in any document). The axis will be empty unless the context node is an element.
Primary node type:	namespace
Implemented	w3c1 , msxml3 , msxml4 , msxml.NET

parent

Description:	Contains the direct parent node (and only the direct parent node) of the context node.
Primary node type:	element
Shorthand:	..
Implemented	w3c1 , msxml2 (only shorthand syntax), msxml26 , msxml3 , msxml4 , msxml.NET

preceding

Description:	Contains all nodes that come before the context node in the document order. This includes all elements that are already closed (their closing tag comes before the context node in the document) and therefore excludes all ancestors of the context node.
Primary node type:	element
Implemented	w3c1 , msxml3 , msxml4 , msxml.NET

preceding-sibling

Description:	Contains all siblings (children of the same parent node) of the context node that come before the context node in document order
Primary node type:	element
Implemented	w3c1 , msxml3 , msxml4 , msxml.NET

self

Description:	Contains only the context node itself.
Primary node type:	element
Shorthand:	.
Implemented	w3c1 , msxml2 (only shorthand syntax) , msxml26 , msxml3 , msxml4 , msxml.NET

Node tests

*

Description:	Returns true for all nodes of the primary type of the axis.
Implemented	w3c1 , msxml2 , msxml26 , msxml3 , msxml4 , msxml.NET

comment()

Description:	Returns true for all comment nodes
Implemented	w3c1 , msxml2 , msxml26 , msxml3 , msxml4 , msxml.NET

literal name

Description:	Returns true for all nodes of that name. If the node test is 'PERSON', it returns true for all nodes of name 'PERSON'
Implemented	w3c1 , msxml2 , msxml26 , msxml3 , msxml4 , msxml.NET

node()

Description:	Returns true for all nodes, except attributes and namespaces
Implemented	w3c1 , msxml2 , msxml26 , msxml3 , msxml4 , msxml.NET

processing-instruction(name?)

Description:	Returns true for all processing instruction nodes. If a name is passed, returns true only for processing instruction nodes of that name
Implemented	w3c1 , msxml2 (called pi() in MSXML2) , msxml26 , msxml3 , msxml4 , msxml.NET

text()

Description:	Returns true for all text nodes
Implemented	w3c1 , msxml2 , msxml26 , msxml3 , msxml4 , msxml.NET

Functions

Each function is described by a line of this form:
return-type **function-name** (parameters)

boolean

Syntax: `boolean = boolean (object)`

Converts anything passed to it to a boolean. `boolean(attribute::name)` will return true if the context node has a name attribute.

Parameters **object** Numbers result in true if they are not zero or NaN. Strings are true if their length is non-zero. Node-sets return true if they are non-empty.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

ceiling

Syntax: `number = ceiling (number)`

Round a passed number to the smallest integer that is not smaller than the passed number. `ceiling(1.1)` returns 2

Parameters **number** The number that must be rounded

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

concat

Syntax: `string = concat (string, string+)`

Concatenates all passed strings to one string. `concat('con', 'c', 'a', 't')` returns concat

Parameters **string** The first string
 string All following strings.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Used in example(s) [Creating listboxes and checkboxes using parameters](#),
[Creating listboxes and checkboxes using variables](#)

contains

Syntax: `boolean = contains (string, string)`

Returns true if string1 contains string2. `contains('Teun Duynstee', 'uy')` returns true.

Parameters	string	The source string
	string	The string that must be searched
Implemented:	w3c1 , msxml26 , msxml3 , msxml4 , msxml.NET	

count

Syntax: `number = count (node-set)`

Returns the number of nodes in the passed `node-set`. `count (child::*[@name])` returns the number of child elements of the context node that have a name attribute.

Parameters	node-set	The <code>node-set</code> that is to be counted
Implemented:	w3c1 , msxml26 , msxml3 , msxml4 , msxml.NET	
Used in example(s)	Combining and intersecting two nodesets , Creating a summary of author sales for a publisher , Using different axes	

false

Syntax: `boolean = false ()`

Always returns false. `starts-with(@name, 'T') = false()`

Implemented:	w3c1 , msxml26 , msxml3 , msxml4 , msxml.NET
---------------------	--

floor

Syntax: `number = floor (number)`

Round a passed number to the largest integer that is not larger than the passed number. `floor (2.9)` returns 2 `floor (-1.1)` returns -2

Parameters	number	The number that must be rounded
Implemented:	w3c1 , msxml3 , msxml4 , msxml.NET	

id

Syntax: `node-set = id (string)`

Returns the element identified by the passed identifier. Note that this will only work in validated documents, because for non-validated documents the parser has no way of knowing which attributes represent ID values.

Parameters	string	The ID value
-------------------	---------------	--------------

Implemented: [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

lang

Syntax: `boolean = lang (string)`

Returns true if the language of the context node is the same as the passed language parameter. The language of the context node can be set using the `xml:lang` attribute on it or any of its ancestors. This feature of XML isn't used frequently. `lang('en')` returns true for English language nodes.

Parameters **string** Language identifier. `lang('en')` returns true for english language nodes.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

last

Syntax: `number = last ()`

Returns the index number of the last node in the current context node-set. `child::*[position() = last()-1]` selects the penultimate child element of the context node.

Implemented: [w3c1](#), [msxml2](#) (called `end()` in MSXML2), [msxml26](#) (does not work when used on the descendant axis), [msxml3](#), [msxml4](#), [msxml.NET](#)

local-name

Syntax: `string = local-name (node-set?)`

Returns the local part of the name of the first node (in document order) in the passed nodeset. The local part of an `xsl:value-of` element is `value-of`.

Parameters **node-set** If no node-set is specified, the current context node is used.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

name

Syntax: `string = name (node-set?)`

Returns the name of the passed node. This is the fully qualified name, including namespace prefix.

Parameters **node-set** If no node-set is specified, the current context node is used.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

namespace-uri

Syntax: `string = namespace-uri (node-set?)`

Returns the full URI that defines the namespace of the passed node. `namespace-uri(@href)` in an XHTML document might return `http://www.w3.org/Profiles/XHTML-transitional`

Parameters **node-set** If no `node-set` is specified, the current context node is used.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

normalize-space

Syntax: `string = normalize-space (string?)`

Returns the whitespace-normalized version of the passed string. This means that all leading and trailing whitespace gets stripped and all sequences of whitespace get combined to one single space.

`normalize-space(' some text ')` would return `some text`

Parameters **string** If no string is passed, the current node is converted to a string.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

not

Syntax: `boolean = not (boolean)`

Returns the inverse of the passed value. `not(@name)` returns `true` if there is no `name` attribute on the context node.

Parameters **boolean**

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

number

Syntax: `number = number (object?)`

Converts parameter to a number. `number(' -3.6 ')` returns the number `-3.6`. The number function does not use any localized settings, so you must only use this conversion when the format of the numeric data is language-neutral.

Parameters **object** If nothing is passed, the current context node is used.

Implemented: [w3c1](#), [msxml26](#) (seems to return a string instead of a number (?)), [msxml3](#), [msxml4](#), [msxml.NET](#)

position

Syntax: `number = position ()`

Returns the position of the current context node in the current context node-set. `position()` returns 1 for the first node in the context node set.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Used in example(s) [Creating a summary of author sales for a publisher](#)

round

Syntax: `number = round (number)`

Round a passed number to the nearest integer. `round(1.5)` returns 2, `round(-1.7)` returns -2

Parameters **number** The number that must be rounded

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

starts-with

Syntax: `boolean = starts-with (string, string)`

Returns true if `string1` starts with `string2`. `starts-with(@name, 'T')` returns true if the value of the name attribute starts with a capital T

Parameters **string** The string that must be checked

string The substring that must be searched

Implemented: [w3c1](#), [msxml26](#) (somehow this fails to work in the test attribute of an `xsl:if` element), [msxml3](#), [msxml4](#), [msxml.NET](#)

string

Syntax: `string = string (object?)`

Converts the passed object to a string value.

Parameters **object** If nothing is passed, the result is an empty string.

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

string-length

Syntax: `number = string-length (string?)`

Returns the number of characters in the passed string. `string-length('Teun Duynstee')` returns 13

Parameters **string** If nothing is passed, the current context is converted to a string.
Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

substring

Syntax: `string = substring (string, number, number?)`

Returns the substring from the passed string starting at the `number1`, with the length of `number2`. If no `number2` is passed, the substring runs to the end of the passed string. `substring('Teun Duynstee', 6)` returns `Duynstee`

Parameters **string** The string that will be used as source for the substring
 number Start location of the substring.
 number Length of the substring.
Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

substring-after

Syntax: `string = substring-after (string, string)`

Returns the string part following the first occurrence of the second passed string inside the first passed string. The return value of `substring-after('2000/3/22', '/')` would be `'3/22'`.

Parameters **string** The string that serves as source
 string The string that is searched in the source string
Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

substring-before

Syntax: `string = substring-before (string, string)`

Returns the string part preceding the first occurrence of the second passed string inside the first passed string. The return value of `substring-before('2000/3/22', '/')` would be `'2000'`.

Parameters **string** The string that serves as source
 string The string that is searched in the source string
Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

sum

Syntax: number = **sum** (node-set)

Sums the values of all nodes in the set when converted to number. `sum(student/@age)` returns the sum of all age attributes on the student elements on the child axis of the context node.

Parameters **node-set** The node-set containing all values to be summed.

Implemented: [w3c1, msxml3, msxml4, msxml.NET](#)

Used in example(s) [Creating a summary of author sales for a publisher](#)

translate

Syntax: string = **translate** (string, string, string)

Translates characters in a string to other characters. Translation pairs are specified by the second and third strings. `translate('A Space Odissei', 'i', 'y')` would result in 'A Space Odyssey'. `translate('abcdefg', 'aceg', 'ACE')` result in 'AbCdEf'. The final g gets translated to nothing, because the third string has no counterpart for that position in the second string.

Parameters **string** String to be translated character by character.

string String defining which characters must be translated.

string String defining what the characters from the second string should be translated to.

Implemented: [w3c1, msxml26, msxml3, msxml4, msxml.NET](#)

Used in example(s) [Creating an HTML document with 'previous' and 'next' links](#)

true

Syntax: boolean = **true** ()

Returns always true.

Implemented: [w3c1, msxml26, msxml3, msxml4, msxml.NET](#)

A few examples of XPath expressions

Select all descending elements from the root

`/descendant::*`

Select ancestor elements of the context node that are named 'Chapter'

```
ancestor::Chapter
```

Select nodes that have more than two direct children of name 'Skip'

```
/descendant::node()[count(child::Skip) < 2]
```

Select all Student elements whose name attribute start with an A (using shorthand notation)

```
//Student[starts-with(@name, 'A')]
```

XSLT Reference

This reference appendix describes the elements, functions and XPath functions which can be used with XSLT (which was covered in Chapter 4.)

Elements

xsl:apply-imports

For calling a template from an imported stylesheet that was overruled in the importing stylesheet. This is normally used if you want to add functionality to a standard template that you imported using `xsl:import`.

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

xsl:apply-templates

Used to pass the context on to another template. The `select` attribute specifies which nodes should be transformed now, the processor decides which templates will be used.

Attributes

select (not required)

Type: node-set-expression

Attribute Value Template: no

mode (not required)

Type: qname

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:sort](#), [xsl:with-param](#)

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s) [Creating a summary of author sales for a publisher](#), [Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#), [Numbering paragraphs and chapters](#), [Using different axes](#), [Whitespace preserving and stripping](#)

xsl:attribute

Generates an attribute in the destination document. It should be used in the context of an element (either a literal, xsl:element or some other element that generates an element in the output). It must occur before any text or element content is generated.

Attributes

name (required)

Type: qname

Attribute Value Template: yes

namespace (not required)

Type: uri-reference

Attribute Value Template: yes

Implemented

[w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

[xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:call-template](#), [xsl:choose](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by

[xsl:attribute-set](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s)

[Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#), [Generating a new stylesheet](#)

xsl:attribute-set

For defining a set of attributes that can be added to an element as a group by specifying the attribute-set name in the use-attribute-sets attribute on the [xsl:element](#) element.

Attributes

name (required)

Type: qname

Attribute Value Template: no

use-attribute-sets (not required)

Type: qnames

Attribute Value Template: no

Implemented

[w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

[xsl:attribute](#)

Can be contained by

[xsl:stylesheet](#), [xsl:transform](#)

xsl:call-template

Calling a template by name. Causes no context switch (change of context node) as apply-templates and for-each do. The template you call by name will still be processing the same context node as your current template. This element can be used to reuse the same functionality in several templates.

Attributes

name (required)

Type: qname

Attribute Value Template: no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:with-param](#)

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s) [Creating an HTML document with 'previous' and 'next' links](#), [Using different axes](#)

xsl:choose

For implementing the choose/when/otherwise construct. Compare to Case/Select in Visual Basic.

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:otherwise](#), [xsl:when](#)

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

xsl:comment

For generating a comment node in the destination document.

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:call-template](#), [xsl:choose](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

xsl:copy

Generates a copy of the context node in the destination document. Does not copy any children or attributes.

Attributes

use-attribute-sets (not required)

Type: qnames

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s) [Creating a summary of author sales for a publisher](#)

xsl:copy-of

Copies a full tree including attributes to the destination document. If multiple nodes are matched by the select attribute, they are all copied. If you have an XML fragment stored in a variable, xsl:copy-of is the handiest element to send the variables content to the output.

Attributes

select (required)

Type: expression

Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s) [Creating an HTML document with 'previous' and 'next' links](#)

xsl:decimal-format

Top-level element for defining settings for conversion to numeric values.

Attributes

name (not required)

Type: qname

Attribute Value Template: no

decimal-separator (not required)

Type: char

Attribute Value Template: no

grouping-separator (not required)

Type: char

Attribute Value Template: no

infinity (not required)

Type: string

Attribute Value Template: no

minus-sign (not required)

Type: char

Attribute Value Template: no

NaN (not required)

Type: string

Attribute Value Template: no

percent (not required)

Type: char

Attribute Value Template: no

per-mille (not required)

Type: char

Attribute Value Template: no

zero-digit (not required)

Type: char

Attribute Value Template: no

digit (not required)

Type: char

Attribute Value Template: no

pattern-separator (not required)

Type: char
Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

xsl:element

Generates an element with the specified name in the destination document.

Attributes

name (required)

Type: qname
Attribute Value Template: yes

namespace (not required)

Type: uri-reference
Attribute Value Template: yes

use-attribute-sets (not required)

Type: qnames
Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

xsl:fallback

Can be used to specify actions to be executed if the action of its parent element is not supported by the processor.

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#),

[xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by

[xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

xsl:for-each

For repeatedly performing a fragment of XSLT elements. The context is shifted to the current node in the loop.

Attributes

select (required)

Type: node-set-expression
Attribute Value Template: no

Implemented

[w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

[xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:sort](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by

[xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s)

[Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#), [Generating a new stylesheet](#)

xsl:if

Executes the contained elements only if the test expression returns true (or a filled node set).

Attributes

test (required)

Type: boolean-expression
Attribute Value Template: no

Implemented

[w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

[xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#),

	xsl:element , xsl:fallback , xsl:for-each , xsl:if , xsl:message , xsl:number , xsl:processing-instruction , xsl:text , xsl:value-of , xsl:variable
Can be contained by	xsl:attribute , xsl:comment , xsl:copy , xsl:element , xsl:fallback , xsl:for-each , xsl:if , xsl:message , xsl:otherwise , xsl:param , xsl:processing-instruction , xsl:template , xsl:variable , xsl:when
Used in example(s)	Creating listboxes and checkboxes using parameters , Creating listboxes and checkboxes using variables

xsl:import

Imports the templates from an external stylesheet document into the current document. The priority of these imported templates is very low, so if a template is implemented for the same pattern, it will always prevail over the imported template. The imported template can be called from the overriding template using [xsl:apply-imports](#).

Attributes

href (required)

Type: uri-reference

Attribute Value Template: no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

xsl:include

Includes templates from an external document as if they were part of the importing document.

Attributes

href (required)

Type: uri-reference

Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

xsl:key

Can be used to create index-like structures that can be queried from the [key](#) function. It basically is a way to describe name/value pairs inside the source document (like a Dictionary object in VB or an associative array in Perl). Only in XSLT, more than one value can be found for one key and the same value can be accessed by multiple keys.

Attributes

name (required)	The name that can be used to refer to this key.
Type:	qname
Attribute Value Template:	no
match (required)	The pattern defines which nodes in the source document can be accessed using this key. In the name/value pair analogy, this would be the definition of the value.
Type:	pattern
Attribute Value Template:	no
use (required)	This expression defines what the key for accessing each value would be. Example: if an element PERSON is matched by the match attribute and the use attribute equals "@name", the key function can be used to find this specific PERSON element by passing the value of its name attribute.
Type:	expression
Attribute Value Template:	no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

Used in example(s) [Creating a summary of author sales for a publisher](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#)

xsl:message

To issue error messages or warnings. The content is the message.

Attributes

terminate (not required)	
Type:	yes no
Attribute Value Template:	no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#),

[xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#),
[xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#),
[xsl:variable](#)

Can be
contained by

[xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#),
[xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#),
[xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

xsl:namespace-alias

Used to make a certain namespace appear in the destination document without using that namespace in the stylesheet. The main use of this element is in generating new XSLT stylesheets.

Attributes

stylesheet-prefix (required)	The prefix for the namespace that is used in the stylesheet
Type:	prefix #default
Attribute Value Template:	no
result-prefix (required)	The prefix for the namespace that must replace the aliased namespace in the destination document.
Type:	prefix #default
Attribute Value Template:	no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be
contained by [xsl:stylesheet](#), [xsl:transform](#)

Used in
example(s) [Generating a new stylesheet](#)

xsl:number

For outputting the number of a paragraph or chapter in a specified format. Has very flexible features, to allow for different numbering rules.

Attributes

level (not required)	The value 'single' counts the location of the nearest node matched by the count attribute (along the ancestor axis) relative to its preceding siblings of the same name. Typical output: chapter number. The value 'multiple' will count the location of the all nodes matched by the count attribute (along the ancestor axis) relative to their preceding siblings of the same name. Typical output: paragraph number of form 4.5.3. The value 'any' will count the location of the nearest node matched by the count attribute (along the ancestor axis) relative to their
-----------------------------	---

Type:	preceding nodes (not only siblings) of the same name. Typical output: bookmark number
Attribute Value Template:	single multiple any
count (not required)	no
Type:	Specifies the type of node that is to be counted
Attribute Value Template:	pattern
from (not required)	no
Type:	Specifies the starting point for counting
Attribute Value Template:	pattern
value (not required)	no
Type:	Used to specify the numeric value directly instead of using 'level', 'count' and 'from'.
Attribute Value Template:	number-expression
format (not required)	no
Type:	How to format the numeric value to a string (1 becomes 1, 2, 3, ...; a becomes a, b, c, ...)
Attribute Value Template:	string
lang (not required)	yes
Type:	Language used for alphabetic numbering
Attribute Value Template:	nmtoken
letter-value (not required)	yes
Type:	Some languages have traditional orders of letters specifically for numbering. These orders are often different from the alphabetic order.
Attribute Value Template:	alphabetic traditional
grouping-separator (not required)	yes
Type:	Character to be used for group separation.
Attribute Value Template:	char
grouping-size (not required)	yes
Type:	Number of digits to be separated. grouping-separator=";" and grouping-size="3" causes: 1;000;000
Attribute Value Template:	number
grouping-size (not required)	yes

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in
example(s) [Numbering paragraphs and chapters](#)

xsl:otherwise

Content is executed if none of the [xsl:when](#) elements is matched.

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#),
[xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#),
[xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#),
[xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#),
[xsl:variable](#)

**Can be
contained by** [xsl:choose](#)

xsl:output

Top level element for setting properties regarding the output style of the destination document. The xsloutput element basically describes how the translation from a created XML tree to a character array (string) happens.

Attributes

method (not required)	xml is default, html will create empty elements like and use HTML entities like à. text will cause no output escaping to happen at all (no entity references in output.)
Type:	xml html text qname-but-not-ncname
Attribute Value Template:	no
version (not required)	
Type:	nmtoken
Attribute Value Template:	no
encoding (not required)	
Type:	string
Attribute Value Template:	no
omit-xml-declaration (not required)	
Type:	yes no
Attribute Value Template:	no

standalone (not required)

Type: yes|no

Attribute Value Template: no

doctype-public (not required)

Type: string

Attribute Value Template: no

doctype-system (not required)

Type: string

Attribute Value Template: no

CDATA-section-elements (not required)

Specifies a list of elements that should have their content escaped by using a CDATA section instead of entities.

Type: qnames

Attribute Value Template: no

indent (not required)

Specifies to addition of extra whitespace for readability

Type: yes|no

Attribute Value Template: no

media-type (not required)

To specify a specific MIME type while writing out content.

Type: string

Attribute Value Template: no

Implemented

[w3c1](#), [msxml26](#) (No support for methods html and text) , [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be

contained by

[xsl:stylesheet](#), [xsl:transform](#)

Used in

example(s)

[Creating a summary of author sales for a publisher](#), [Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#)

xsl:param

Defines a parameter in a [xsl:template](#) or [xsl:stylesheet](#) .

Attributes

name (required)

Type: QName

Attribute Value Template: no

Specifies the default value for the parameter

select (not required)

Type: expression

Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

Used in example(s) [Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#)

xsl:preserve-space

Allows you to define which elements in the source document should have their whitespace content preserved. See [xsl:strip-space](#).

Attributes

elements (required)

Type: tokens

Attribute Value Template: no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

Used in example(s) [Whitespace preserving and stripping](#)

xsl:processing-instruction

Generate a processing instruction in the destination document.

Attributes

name (required)

Type: ncname

Attribute Value Template: yes

Implemented [w3c1](#), [msxml2](#) (Caution: the xsl:processing-instruction element is called xsl:pi in IE5), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:call-template](#), [xsl:choose](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:fallback](#), [xsl:for-each](#),

[xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:text](#), [xsl:value-of](#),
[xsl:variable](#)

Can be contained by [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#),
[xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:template](#), [xsl:variable](#),
[xsl:when](#)

xsl:sort

Allows specifying a sort order for [xsl:apply-templates](#) and [xsl:for-each](#) elements. Multiple sort elements can be specified for primary and secondary sorting keys.

Attributes

select (not required)

Type: string-expression

Attribute Value Template: no

lang (not required)

Type: nmtoken

Attribute Value Template: yes

data-type (not required)

Type: text|number|qname-but-not-ncname

Attribute Value Template: yes

order (not required)

Type: ascending|descending

Attribute Value Template: yes

case-order (not required) Note that case insensitive sorting is not supported

Type: upper-first|lower-first

Attribute Value Template: yes

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:apply-templates](#), [xsl:for-each](#)

Used in example(s) [Creating a summary of author sales for a publisher](#)

xsl:strip-space

Allows you to define which elements in the source document should have their whitespace content stripped. See [xsl:preserve-space](#).

Attributes

elements (required)

Type: tokens

Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

Used in example(s) [Whitespace preserving and stripping](#)

xsl:stylesheet

The root element for a stylesheet. Synonym to [xsl:transform](#).

Attributes

id (not required)

Type: id

Attribute Value Template: no

extension–element–prefixes (not required) Allows you to specify which namespace prefixes are XSLT extension namespaces (like msxml)

Type: tokens

Attribute Value Template: no

exclude–result–prefixes (not required) Namespaces that are only relevant in the stylesheet or in the source document, but not in the result document, can be removed from the output by specifying them here.

Type: tokens

Attribute Value Template: no

version (required)

Type: number

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:attribute-set](#), [xsl:decimal-format](#), [xsl:import](#), [xsl:include](#), [xsl:key](#), [xsl:namespace-alias](#), [xsl:output](#), [xsl:param](#), [xsl:preserve-space](#), [xsl:strip-space](#), [xsl:template](#), [xsl:variable](#)

Can be contained by

Used in example(s) [Combining and intersecting two nodesets](#), [Creating a summary of author sales for a publisher](#), [Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes](#)

[using parameters](#), [Creating listboxes and checkboxes using variables](#), [Generating a new stylesheet](#), [Numbering paragraphs and chapters](#), [Using different axes](#), [Whitespace preserving and stripping](#)

xsl:template

Defines a transformation rule. Some templates are built-in and don't have to be defined.

Attributes

match (not required)

Type: pattern

Attribute Value Template: no

name (not required)

Type: qname

Attribute Value Template: no

priority (not required)

Type: number

Attribute Value Template: no

mode (not required)

Type: qname

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#) (except for the mode attribute), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by [xsl:stylesheet](#), [xsl:transform](#)

Used in example(s) [Combining and intersecting two nodesets](#), [Creating a summary of author sales for a publisher](#), [Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#), [Generating a new stylesheet](#), [Numbering paragraphs and chapters](#), [Using different axes](#), [Whitespace preserving and stripping](#)

xsl:text

Generates a text string from it's content. Whitespace is never stripped from a text element.

Attributes

disable-output-escaping (not required)

Type: yes|no

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)

Used in example(s) [Creating a summary of author sales for a publisher](#), [Creating an HTML document with 'previous' and 'next' links](#)

xsl:transform

Identical to [xsl:stylesheet](#)

Attributes

id (not required)

Type: id

Attribute Value Template: no

extension-element-prefixes (not required)

Type: tokens

Attribute Value Template: no

exclude-result-prefixes (not required)

Type: tokens

Attribute Value Template: no

version (required)

Type: number

Attribute Value Template: no

Implemented [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:attribute-set](#), [xsl:decimal-format](#), [xsl:import](#), [xsl:include](#), [xsl:key](#), [xsl:namespace-alias](#), [xsl:output](#), [xsl:param](#), [xsl:preserve-space](#), [xsl:strip-space](#), [xsl:template](#), [xsl:variable](#)

Can be contained by

xsl:value-of

Generates a text string with the value of the select expression.

Attributes

select (required)

Type: string-expression

Attribute Value Template: no

disable-output-escaping (not required)

You can use this to output < instead of < to the destination document. Note that this will cause your destination to become invalid XML. Normally used to generate HTML or text files.

Type: yes|no

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be

contained by

[xsl:attribute](#), [xsl:comment](#), [xsl:copy](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:otherwise](#), [xsl:param](#), [xsl:processing-instruction](#), [xsl:template](#), [xsl:variable](#), [xsl:when](#)
[Combining and intersecting two nodesets](#), [Creating a summary of author sales for a publisher](#), [Creating an HTML document with 'previous' and 'next' links](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#), [Generating a new stylesheet](#), [Numbering paragraphs and chapters](#), [Using different axes](#), [Whitespace preserving and stripping](#)

Used in

example(s)

xsl:variable

Defines a variable with a value. A variable really is not variable, but constant.

Attributes

name (required)

Type: QName

Attribute Value Template: no

select (not required)

Type: expression

Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

[xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#),

	xsl:element , xsl:fallback , xsl:for-each , xsl:if , xsl:message , xsl:number , xsl:processing-instruction , xsl:text , xsl:value-of , xsl:variable
Can be contained by	xsl:attribute , xsl:comment , xsl:copy , xsl:element , xsl:fallback , xsl:for-each , xsl:if , xsl:message , xsl:otherwise , xsl:param , xsl:processing-instruction , xsl:stylesheet , xsl:template , xsl:transform , xsl:variable , xsl:when
Used in example(s)	Combining and intersecting two nodesets , Creating listboxes and checkboxes using variables

xsl:when

Represents one of the options for execution in a [xsl:choose](#) block.

Attributes

test (required)

Type: boolean-expression

Attribute Value Template: no

Implemented [w3c1](#), [msxml2](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain [xsl:apply-imports](#), [xsl:apply-templates](#), [xsl:attribute](#), [xsl:call-template](#), [xsl:choose](#), [xsl:comment](#), [xsl:copy](#), [xsl:copy-of](#), [xsl:element](#), [xsl:fallback](#), [xsl:for-each](#), [xsl:if](#), [xsl:message](#), [xsl:number](#), [xsl:processing-instruction](#), [xsl:text](#), [xsl:value-of](#), [xsl:variable](#)

Can be contained by [xsl:choose](#)

xsl:with-param

Defines a parameter on a [xsl:template](#) or [xsl:stylesheet](#) . Also specifies a default value.

Attributes

name (required)

Type: QName

Attribute Value Template: no

select (not required)

Type: expression

Attribute Value Template: no

Implemented [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Can contain

Can be contained by [xsl:apply-templates](#), [xsl:call-template](#)

Used in
example(s) [Creating listboxes and checkboxes using parameters](#)

Functions

The functions are listed, with a brief description and a list of parameters. (A question mark after the parameter signifies that it is optional.) All of these functions are implemented in W3C 1.0.

current

Syntax: `node-set = current ()`

Returns the current context, outside the current expression. For MSXML2 you can use the `context()` function as a workaround. `context(-1)` is synonymous to `current()`

Implemented: [w3c1](#), [msxml26](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

document

Syntax: `node-set = document (object , node-set?)`

Allows to get a reference to an external source document.

Parameters

object	If of type String, the URL of the document to be retrieved, if a node-set, all nodes are converted to strings and all these URLs are retrieved in a node-set.
node-set	Represents the base URL from where relative URLs are resolved.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

element-available

Syntax: `boolean = element-available (string)`

To query availability of a certain extension element.

Parameters

string	Name of the extension element.
---------------	--------------------------------

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

format-number

Syntax: `string = format-number (number , string , string?)`

Formats a numeric value into a formatted and localized string.

Parameters

number	The numeric value to be represented.
---------------	--------------------------------------

- string** The format string that should be used for the formatting.
- string** Reference to a [xsl:decimal-format](#) element to indicate localisation parameters.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

function-available

Syntax: `boolean = function-available (string)`

To query availability of a certain extension function.

Parameters **string** Name of the extension function.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

generate-id

Syntax: `node-set = generate-id (node-set?)`

Generates a unique identifier for the specified node. Each node will cause a different ID, but the same node will always generate the same ID. You cannot be sure that the IDs generated for the a document during multiple transformations will remain identical.

Parameters **node-set** The first node of the passed node-set is used. If no node-set is passed, the current context is used.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

key

Syntax: `node-set = key (string, object)`

Allows to get a reference to a node using the specified [xsl:key](#) .

Parameters **string** The name of the referenced [xsl:key](#) .

object If of type String, the index string for the key. If of type node-set, all nodes are converted to strings and all are used to get nodes back from the key.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Used in example(s) [Creating a summary of author sales for a publisher](#), [Creating listboxes and checkboxes using parameters](#), [Creating listboxes and checkboxes using variables](#)

system-property

Syntax: object = **system-property** (string)

Allows to get certain system properties from the processor.

Parameters **string** The name of the system property. Properties that are always available are xsl:version, xsl:vendor and xsl:vendor-url.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

unparsed-entity-url

Syntax: node-set = **unparsed-entity-url** (string)

Returns the URI of the unparsed entity with the passed name.

Parameters **string** Name of the unparsed entity.

Implemented: [w3c1](#), [msxml3](#), [msxml4](#), [msxml.NET](#)

Inherited XPath Functions

Check the XPath reference for information on these XPath functions. They can all be used in XSLT:

[boolean](#), [ceiling](#), [concat](#), [contains](#), [count](#), [false](#), [floor](#), [id](#), [lang](#), [last](#), [local-name](#), [name](#), [namespace-uri](#), [normalize-space](#), [not](#), [number](#), [position](#), [round](#), [starts-with](#), [string](#), [string-length](#), [substring](#), [substring-after](#), [substring-before](#), [sum](#), [translate](#), [true](#)

Types

boolean	Can have values true and false
char	A single character
expression	A string value, containing an XPath expression.
id	A string value. Must be an XML name. The string value can be used only once as an id in any document.
language-name	A string containing one of the defined language identifiers. American English = EN-US
name	A string value that conforms to the name conventions of XML. That means: no whitespace, should start with either a letter or an _.
names	Multiple name values separated by whitespace.
namespace-prefix	Any string that is defined as a prefix for a namespace.
nname	A name value that does not contain a colon.
node	A node in an XML document. Can be of several types, including: element, attribute, comment, processing instruction, text node, etc...

node-set	A set of nodes in a specific order. Can be of any length.
node-set-expression	A string value, containing an XPath expression that returns nodes.
number	A numeric value. Can be both floating point or integer
object	Anything. Can be a string, a node, a node-set, anything
qname	Qualified name: the full name of a node. Made up of two parts: the local name and the namespace identifier.
qnames	A set of qname values, separated by whitespace.
string	A string value
token	A string value that contains no whitespace.
tokens	Multiple token values separated by whitespace.
uri-reference	Any string that conforms to the URI specification.

Worked code samples

Example: Combining and intersecting two nodesets

This example shows how you can compare and combine the results in different node-sets. Using the union operator (|) two or more sets can be combined to one set (the nodes that occur in both sets should occur only once in the resulting combination). Using this combining of node-sets, we can also find the intersection of two node-sets: the technique to do this was presented by Michael Kay and is shown in this example.

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:variable](#) [xsl:value-of](#)
XSLT and XPath functions used: [count](#)
XML source:

```
<?xml version="1.0"?>
<publisher>
<books>
<book>
  <title>Stranger in a strange land</title>
  <ISBN>0441788386</ISBN>
  <author-ref ref="rh"/>
  <sold>230000</sold>
</book>

<book>
  <title>Starman Jones</title>
  <ISBN>0345328116 </ISBN>
  <author-ref ref="rh"/>
  <author-ref ref="jldr"/>
  <sold>80000</sold>
</book>

<book>
  <title>The Space Merchants</title>
  <ISBN>02352123456 </ISBN>
  <author-ref ref="fp"/>
  <author-ref ref="ck"/>

```

```
<sold>120000</sold>
</book>

</books>
<authors>
  <author id="rh">
    <first_name>Robert</first_name>
    <last_name>Heinlein</last_name>
  </author>
  <author id="ck">
    <first_name>Cyril</first_name>
    <last_name>Kornbluth</last_name>
  </author>
  <author id="fp">
    <first_name>Frederick</first_name>
    <last_name>Pohl</last_name>
  </author>
  <author id="jldr">
    <first_name>Judy-Lyn</first_name>
    <last_name>Del Rey</last_name>
  </author>
</authors>
</publisher>
```

XSLT code:

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

<xsl:variable name="books_by_rh"
  select="//book[author-ref/@ref='rh']"/>
<xsl:variable name="books_by_more_than_one_author"
  select="//book[count(author-ref) > 1]"/>

<xsl:template match="/">
  Total number of books by Robert Heinlein: <xsl:value-of
    select="count($books_by_rh)" />
  Total number of books by several authors: <xsl:value-of
    select="count($books_by_more_than_one_author)" />

  Number of books in the union of the two sets
  <xsl:value-of
    select="count($books_by_rh|$books_by_more_than_one_author)"/>
  Number of books in the intersection of the two sets
  <xsl:value-of
    select="count(
      $books_by_rh[count(.|$books_by_more_than_one_author)=
        count($books_by_more_than_one_author)]
    )"/>
</xsl:template>

</xsl:stylesheet>
```

Result code:

```
<?xml version="1.0" encoding="utf-8"?>
  Total number of books by Robert Heinlein: 2
  Total number of books by several authors: 2

  Number of books in the union of the two sets 3
  Number of books in the intersection of the two sets 1
```

Example: Creating a summary of author sales for a publisher

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:text](#) [xsl:apply-templates](#) [xsl:key](#) [xsl:output](#) [xsl:sort](#) [xsl:copy](#) [xsl:value-of](#)

XSLT and XPath functions used: [key](#) [sum](#) [count](#) [position](#)

XML source:

```
<?xml version="1.0"?>
<publisher>
<books>
<book>
  <title>Stranger in a strange land</title>
  <ISBN>0441788386</ISBN>
  <author-ref ref="rh"/>
  <sold>230000</sold>
</book>

<book>
  <title>Starman Jones</title>
  <ISBN>0345328116 </ISBN>
  <author-ref ref="rh"/>
  <author-ref ref="jldr"/>
  <sold>80000</sold>
</book>
</books>
<authors>
  <author id="rh">
    <first_name>Robert</first_name>
    <last_name>Heinlein</last_name>
  </author>
  <author id="jldr">
    <first_name>Judy-Lyn</first_name>
    <last_name>Del Rey</last_name>
  </author>
</authors>
</publisher>
```

XSLT code:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
version="1.0">

<xsl:key match="/publisher/books/book"
  use="author-ref/@ref" name="books-by-author"/>
<xsl:output indent="yes" encoding="UTF-16"/>
<xsl:template match="/"><bestsellers-list>
<xsl:apply-templates select="/publisher/authors/author">
  <xsl:sort data-type="number" order="descending"
    select="sum(key('books-by-author', @id)/sold)"/>
</xsl:apply-templates>
</bestsellers-list></xsl:template>

<xsl:template match="author">
<xsl:copy>
<name><xsl:value-of select="last_name"/>,<xsl:text> </xsl:text>
  <xsl:value-of select="first_name"/>
</name>
<total_publications>
  <xsl:value-of select="count(key('books-by-author',
    @id))"/>
</total_publications>
<total_sold>
  <xsl:value-of select="sum(key('books-by-author',
    @id)/sold)"/>
</total_sold>
<rank><xsl:value-of select="position()"/></rank>
</xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Result code:

```
<?xml version="1.0" encoding="UTF-16"?>
<bestsellers-list>
  <author>
    <name>Heinlein, Robert</name>
    <total_publications>2</total_publications>
    <total_sold>310000</total_sold>
    <rank>1</rank>
  </author>
  <author>
    <name>Del Rey, Judy-Lyn</name>
    <total_publications>1</total_publications>
    <total_sold>80000</total_sold>
    <rank>2</rank>
  </author>
</bestsellers-list>
```

Example: Creating an HTML document with 'previous' and 'next' links

Creating HTML links in a document that is generated from XML: a recurring task. At best, we would like to have a maintainable way to create links in the result document. Especially internal links (where both the link and the links target are generated from the same source) can be tedious to maintain manually. If you set things up as swonw in this sample, both the links and the anchors (the `` tag) are created using the same code. This guarantees that internal links are always correct.

The last 5 templates in this stylesheet could be used as an import stylesheet in other transformations. The logic has been split up in multiple templates: by doing this, we can import the templates and override one of them to create a slightly different linking behavior (i.e. change only the way to generate a unique identifier to another algorithm).

In this source document, the id attributes are all unique strings, which makes them suitable for use as anchor. The only problem is that they contain spaces and possibly also pound signs (#). These characters are not allowed in the anchor name, so we use the translate function to change all spaces and #'s to underscores (_). Because this template is used for both creating the links and creating the anchors, we can't go wrong!

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:apply-templates](#) [xsl:for-each](#) [xsl:value-of](#) [xsl:call-template](#) [xsl:copy-of](#) [xsl:param](#) [xsl:attribute](#) [xsl:text](#) [xsl:output](#)

XSLT and XPath functions used: [translate](#)

XML source:

```
<?xml version="1.0"?>
<book id="book oos" title="The origin of species">
  <part id="oos part 1" title="Opening chapters">
    <chapter id="oos ch 1" title="An historical sketch">
      <par id="par 1">
        text text text
      </par>
      <par id="par 2">
        text text text text text
      </par>
      <par id="par 3">
        text text text text text text text text text
      </par>
    </chapter>
    <chapter id="oos ch 2" title="Introduction">
      <par id="par 4">
        text text text text text
      </par>
      <par id="par 4">
        <b>text text</b> text text text
      </par>
      <par id="par 5">
        text text text text text
      </par>
      <par id="par 6">
        text text text text text
      </par>
    </chapter>
  </part>
  <part id="oos part 2" title="Later chapters">
```

```
<chapter id="oos ch 3" title="Variation under domestication">
  <par id="par 7">
    text text text text text
  </par>
  <par id="par 8">
    text text text text text
  </par>
</chapter>
<chapter id="oos ch 4" title="Variation under nature">
  <par id="par 9">
    text text text text text
  </par>
</chapter>
<chapter id="oos ch 5" title="Struggle for existence">
  <par id="par 10">
    text text text text text
  </par>
  <par id="par 11">
    text text text text text
  </par>
  <par id="par 12">
    text text text text text
  </par>
</chapter>
<chapter id="oos ch 6" title="Natural selection">
  <par id="par 13">
    text text text text text
  </par>
</chapter>
<chapter id="oos ch 7" title="Laws of variation">
  <par id="par 14">
    text text text text text
  </par>
</chapter>
</part>

</book>
```

**XSLT
code:**

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

  <xsl:template match="/book">
    <html><body>
      <xsl:call-template name="anchor-tag-for"/>
      <h1><xsl:value-of select="@title"/></h1>
      <xsl:apply-templates select="part"/>
    </body></html>
  </xsl:template>

  <xsl:template match="part">
    <xsl:call-template name="anchor-tag-for"/>
    <h2><xsl:value-of select="@title"/></h2>
    <xsl:apply-templates select="chapter"/>
  </xsl:template>
```

```
<xsl:template match="chapter">
  <xsl:for-each select="preceding-sibling::chapter[1]">
    Prev: <xsl:call-template name="link-to"/><br/>
  </xsl:for-each>
  <xsl:for-each select="following-sibling::chapter[1]">
    Next: <xsl:call-template name="link-to"/><br/>
  </xsl:for-each>
  <xsl:call-template name="anchor-tag-for"/>
  <h3><xsl:value-of select="@title"/></h3>
  <xsl:apply-templates select="par"/>
</xsl:template>

<xsl:template match="par">
  <p>
    <xsl:call-template name="anchor-tag-for"/>
    <xsl:copy-of select="node()"/>
  </p>
</xsl:template>

<!-- It is important to have a way to reference to a specific location in the
source document with a string value. This can be used to create links to
a specific location in the content and much more. In this case, every
node we want to use as a link has a unique attribute called id. -->
<xsl:template name="node-identifier">
  <xsl:param name="id-node" select="self:*"/>
  <xsl:value-of select="translate($id-node/@id, ' #', '___')"/>
</xsl:template>

<!-- Create an HTML link to any section in the document. A link should always
be the same, both from the index or from another chapter. We locate the
logic of link-creation in this template. It relies on other templates for
determining the file location and the need for an anchor string
(trailing #blabla) -->
<xsl:template name="link-to">
  <xsl:param name="linktext" select="@title"/>
  <a>
    <xsl:attribute name="href">
      <!-- first the filename -->
      <xsl:call-template name="file-of"/>
      <xsl:text>#</xsl:text>
      <xsl:call-template name="anchor-of"/>
    </xsl:attribute>
    <xsl:value-of select="$linktext"/>
  </a>
</xsl:template>

<!-- For creating links, one must know the filename. Calculating the filename
is isolated in this template. In later versions, when we distribute the
content over many files, we will overrule this template with much more
interesting versions. -->
<xsl:template name="file-of">
  <xsl:text>doc.htm</xsl:text>
</xsl:template>

<!-- Here we decide if we need a trailing hash-string to identify the exact
```

```
    spot in a page where to find a certain piece of information. For now we have
    only one file, so we'll always need this extra info. -->
<xsl:template name="anchor-of">
  <xsl:call-template name="node-identifier"/>
</xsl:template>

<xsl:template name="anchor-tag-for">
  <xsl:param name="anchor-node" select="self::*"/>
  <xsl:for-each select="$anchor-node">
    <a>
      <xsl:attribute name="name">
        <xsl:call-template name="anchor-of"/>
      </xsl:attribute>
    </a>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

**Result
code:**

```
<?xml version="1.0" encoding="utf-8"?>
<html><body><a name="book_oos"/>
  <h1>The origin of species</h1>
  <a name="oos_part_1"/>
  <h2>Opening chapters</h2>
  Next: <a href="doc.htm#oos_ch_2">Introduction</a><br/>
  <a name="oos_ch_1"/>
  <h3>An historical sketch</h3>
  <p><a name="par_1"/>
    text text text
  </p>
  <p><a name="par_2"/>
    text text text text text
  </p>
  <p><a name="par_3"/>
    text text text text text text text text text
  </p>
  Prev: <a href="doc.htm#oos_ch_1">An historical sketch</a><br/>
  <a name="oos_ch_2"/>
  <h3>Introduction</h3>
  <p><a name="par_4"/>
    text text text text text
  </p>
  <p><a name="par_4"/>
    <b>text text</b> text text text
  </p>
  <p><a name="par_5"/>
    text text text text text
  </p>
  <p><a name="par_6"/>
    text text text text text
  </p>
  <a name="oos_part_2"/>
  <h2>Later chapters</h2>
```

```
Next: <a href="doc.htm#oos_ch_4">Variation under nature</a><br/>
<a name="oos_ch_3"/>
<h3>Variation under domestication</h3>
<p><a name="par_7"/>
    text text text text text
</p>
<p><a name="par_8"/>
    text text text text text
</p>
Prev: <a href="doc.htm#oos_ch_3">Variation under domestication</a><br/>
Next: <a href="doc.htm#oos_ch_5">Struggle for existence</a><br/>
<a name="oos_ch_4"/>
<h3>Variation under nature</h3>
<p><a name="par_9"/>
    text text text text text
</p>
Prev: <a href="doc.htm#oos_ch_4">Variation under nature</a><br/>
Next: <a href="doc.htm#oos_ch_6">Natural selection</a><br/>
<a name="oos_ch_5"/>
<h3>Struggle for existence</h3>
<p><a name="par_10"/>
    text text text text text
</p>
<p><a name="par_11"/>
    text text text text text
</p>
<p><a name="par_12"/>
    text text text text text
</p>
Prev: <a href="doc.htm#oos_ch_5">Struggle for existence</a><br/>
Next: <a href="doc.htm#oos_ch_7">Laws of variation</a><br/>
<a name="oos_ch_6"/>
<h3>Natural selection</h3>
<p><a name="par_13"/>
    text text text text text
</p>
Prev: <a href="doc.htm#oos_ch_6">Natural selection</a><br/>
<a name="oos_ch_7"/>
<h3>Laws of variation</h3>
<p><a name="par_14"/>
    text text text text text
</p>
</body></html>
```

Example: Creating listboxes and checkboxes using parameters

Creating listboxes and sets of checkboxes from XML data is a recurring task. In these situations, often the possible values are stored at one location in the source document, while the chosen values (which must be represented by adding 'SELECTED' or 'CHECKED' to the corresponding item) are probably stored in another place or even another document.

In this example we use passing parameters as a way to match stored values against possible values. The key element is used to have easy access to lists of possible values.

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:apply-templates](#) [xsl:value-of](#) [xsl:key](#) [xsl:output](#) [xsl:param](#) [xsl:with-param](#) [xsl:if](#) [xsl:for-each](#) [xsl:attribute](#)

XSLT and XPath functions used: [key](#) [concat](#)

XML source:

```
<?xml version="1.0"?>
<questionnaire>
  <questions>
    <question id="1" listref="agree" style="listbox">
      <text>As the global second and third events in
order of spectators, the World and European
championships soccer deserve more coverage
on international news site s.a. CNN.com.
</text>
      <value>4</value>
    </question>
    <question id="2" listref="colors" style="checkbox">
      <text>What are your favorite colors?</text>
      <value>2</value>
      <value>4</value>
    </question>
  </questions>
  <answer-lists>
    <list name="colors">
      <option id="1" name="red" />
      <option id="2" name="yellow" />
      <option id="3" name="green" />
      <option id="4" name="red" />
      <option id="5" name="red" />
    </list>
    <list name="agree">
      <option id="1" name="strongly disagree" />
      <option id="2" name="disagree" />
      <option id="3" name="not sure" />
      <option id="4" name="agree" />
      <option id="5" name="strongly agree" />
    </list>
  </answer-lists>
</questionnaire>
```

XSLT code:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key name="lists" match="//list" use="attribute::name"/>
  <xsl:output method="xml" indent="yes" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <HTML>
      <HEAD/>
      <BODY>
        <FORM>
          <xsl:apply-templates
```

```
        select="questionnaire/questions/question"/>
    </FORM>
</BODY>
</HTML>
</xsl:template>

<xsl:template match="question">
<P>
<xsl:value-of select="child::text/text()"/>
<xsl:apply-templates select="key('lists', @listref)">
    <xsl:with-param name="selected-values" select="value"/>
    <xsl:with-param name="style" select="@style"/>
    <xsl:with-param name="question_id" select="concat('q_',@id)"/>
</xsl:apply-templates>

</P>
</xsl:template>

<xsl:template match="list">
    <xsl:param name="selected-values"/>
    <xsl:param name="style">listbox</xsl:param>
    <xsl:param name="question_id"/>

    <xsl:if test="$style='checkbox'">
        <xsl:for-each select="option">
            <BR/>
            <INPUT TYPE="checkbox" >
                <xsl:attribute name="name">
                    <xsl:value-of select="$question_id"/>
                </xsl:attribute>
                <xsl:if test="$selected-values/text() = attribute::id">
                    <xsl:attribute name="CHECKED"/>
                </xsl:if>
            </INPUT>
            <xsl:value-of select="@name"/>
        </xsl:for-each>

    </xsl:if>

    <xsl:if test="$style='listbox'">
        <BR/>
        <SELECT >
            <xsl:attribute name="name">
                <xsl:value-of select="$question_id"/>
            </xsl:attribute>
            <xsl:for-each select="option">
                <OPTION>
                    <xsl:if test="$selected-values/text() =
                        attribute::id">
                        <xsl:attribute name="SELECTED"/>
                    </xsl:if>
                    <xsl:attribute name="value">
                        <xsl:value-of select="@id"/>
                    </xsl:attribute>
                    <xsl:value-of select="@name"/>
                </OPTION>
            </xsl:for-each>
        </SELECT>
    </xsl:if>
</xsl:template>
```

```
        </OPTION>
      </xsl:for-each>
    </SELECT>

    </xsl:if>
  </xsl:template>
</xsl:stylesheet>

<HTML>
  <HEAD/>
  <BODY>
    <FORM>
      <P>As the global second and third events in
        order of spectators, the World and European
        championships soccer deserve more coverage on
        international news site s.a. CNN.com.
      <BR/>
      <SELECT name="q_1">
        <OPTION value="1">strongly disagree</OPTION>
        <OPTION value="2">disagree</OPTION>
        <OPTION value="3">not sure</OPTION>
        <OPTION SELECTED="" value="4">agree</OPTION>
        <OPTION value="5">strongly agree</OPTION>
      </SELECT>
    </P>
    <P>What are your favorite colors?<BR/>
    <INPUT TYPE="checkbox" name="q_2"/>red<BR/>
    <INPUT TYPE="checkbox" name="q_2" CHECKED="" />yellow<BR/>
    <INPUT TYPE="checkbox" name="q_2"/>green<BR/>
    <INPUT TYPE="checkbox" name="q_2" CHECKED="" />red<BR/>
    <INPUT TYPE="checkbox" name="q_2"/>red</P>
  </FORM>
</BODY>
</HTML>
```

Result code:

Example: Creating listboxes and checkboxes using variables

Creating listboxes and sets of checkboxes from XML data is a recurring task. In these situations, often the possible values are stored at one location in the source document, while the chosen values (which must be represented by adding 'SELECTED' or 'CHECKED' to the corresponding item) are probably stored in another place or even another document.

In this example we use setting variables as a way to match stored values against possible values inside a for-each loop. The key element is used to have easy access to lists of possible values.

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:apply-templates](#) [xsl:value-of](#) [xsl:key](#) [xsl:output](#) [xsl:variable](#) [xsl:if](#) [xsl:for-each](#) [xsl:attribute](#)

XSLT and XPath functions [key](#) [concat](#)

used:
XML
source:

```
<?xml version="1.0"?>
<questionnaire>
  <questions>
    <question id="1" listref="agree" style="listbox">
      <text>As the global second and third events in
order of spectators, the World and European
championships soccer deserve more coverage
on international news site s.a. CNN.com.
</text>
      <value>4</value>
    </question>
    <question id="2" listref="colors" style="checkbox">
      <text>What are your favorite colors?</text>
      <value>2</value>
      <value>4</value>
    </question>
  </questions>
  <answer-lists>
    <list name="colors">
      <option id="1" name="red" />
      <option id="2" name="yellow" />
      <option id="3" name="green" />
      <option id="4" name="red" />
      <option id="5" name="red" />
    </list>
    <list name="agree">
      <option id="1" name="strongly disagree" />
      <option id="2" name="disagree" />
      <option id="3" name="not sure" />
      <option id="4" name="agree" />
      <option id="5" name="strongly agree" />
    </list>
  </answer-lists>
</questionnaire>
```

XSLT
code:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key name="lists" match="//list" use="attribute::name"/>
  <xsl:output method="html" indent="yes" encoding="ISO-8859-1"/>
  <xsl:template match="/">
    <HTML>
      <HEAD/>
      <BODY>
        <FORM>
          <xsl:apply-templates
            select="questionnaire/questions/question"/>
        </FORM>
      </BODY>
    </HTML>
  </xsl:template>
```

```
<xsl:template match="question">
  <P>
    <xsl:variable name="selected-values" select="value"/>
    <xsl:variable name="style" select="@style"/>
    <xsl:variable name="question_id" select="concat('q_',@id)"/>

    <xsl:value-of select="child::text/text()"/>
    <xsl:for-each select="key('lists', @listref)">
      <xsl:if test="$style='checkbox'">
        <xsl:for-each select="option">
          <BR/>
          <INPUT TYPE="checkbox" >
            <xsl:attribute name="name">
              <xsl:value-of select="$question_id"/>
            </xsl:attribute>
            <xsl:if test="$selected-values/text() = attribute::id">
              <xsl:attribute name="CHECKED"/>
            </xsl:if>
          </INPUT>
          <xsl:value-of select="@name"/>
        </xsl:for-each>

      </xsl:if>

      <xsl:if test="$style='listbox'">
        <BR/>
        <SELECT >
          <xsl:attribute name="name">
            <xsl:value-of select="$question_id"/>
          </xsl:attribute>
          <xsl:for-each select="option">
            <OPTION>
              <xsl:if test="$selected-values/text() = attribute::id">
                <xsl:attribute name="SELECTED"/>
              </xsl:if>
              <xsl:attribute name="value">
                <xsl:value-of select="@id"/>
              </xsl:attribute>
              <xsl:value-of select="@name"/>
            </OPTION>
          </xsl:for-each>
        </SELECT>

      </xsl:if>

    </xsl:for-each>

  </P>
</xsl:template>

</xsl:stylesheet>
```

**Result
code:**

<HTML>

```
<HEAD>
  <meta http-equiv="Content-Type" content="text/html;
    charset=ISO-8859-1">
</HEAD>
<BODY>
  <FORM>
    <P>As the global second and third events in
      order of spectators, the World and European
      championships soccer deserve more coverage on
      international news site s.a. CNN.com.
    <BR><SELECT name="q_1">
      <OPTION value="1">strongly disagree</OPTION>
      <OPTION value="2">disagree</OPTION>
      <OPTION value="3">not sure</OPTION>
      <OPTION SELECTED="" value="4">agree</OPTION>
      <OPTION value="5">strongly agree</OPTION>
    </SELECT>
    </P>
    <P>What are your favorite colors?<BR><INPUT
      TYPE="checkbox" name="q_2">red<BR><INPUT
      TYPE="checkbox" name="q_2" CHECKED="">yellow<BR><INPUT
      TYPE="checkbox" name="q_2">green<BR><INPUT
      TYPE="checkbox" name="q_2" CHECKED="">red<BR><INPUT
      TYPE="checkbox" name="q_2">red
    </P>
  </FORM>
</BODY>
</HTML>
```

Example: Generating a new stylesheet

As XSLT stylesheets are formulated in XML and the result of an XSLT transformation can be any XML document, wouldn't it be interesting to generate stylesheets on the fly using another XSLT stylesheet? But how would you do that? How do you output an `xsl:template` element without causing syntax errors in the first stylesheet? After all, a template cannot hold another template. This is where the slightly obscure element `xsl:namespace-alias` comes to the rescue. It allows us to use elements of a namespace A in our stylesheet, but just before the result tree is written to the result document, all nodes with namespace a are changed to another namespace (B).

When you try to generate a stylesheet, you would just create elements using a dummy namespace (`dummy:template`). Then, using `xsl:namespace-alias`, we would specify the in the output, this dummy namespace should be converted to the XSLT namespace (`http://www.w3.org/1999/XSL/Transform`).

In this example, we generate a stylesheet that can transform a piece of code like 2231971 to something like

where the names of the months come from the source document of the first transformation. This

allows you to keep even some of the content bits that appear inside the stylesheet in a separate XML document, merging them as you need them. In this case, we used the spanish content, but we may have many.

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:value-of](#) [xsl:namespace-alias](#) [xsl:attribute](#) [xsl:for-each](#)

XML source:

```
<?xml version="1.0"?>
  <months language="es">
```

```
<month number="1">enero</month>
<month number="2">febrero</month>
<month number="3">marcho</month>
<month number="4">avril</month>
<month number="5">mayo</month>
<month number="6">junio</month>
<month number="7">julio</month>
<month number="8">agosto</month>
<month number="9">setembre</month>
<month number="10">octubre</month>
<month number="11">noviembre</month>
<month number="12">diciembre</month>
</months>
```

XSLT code:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0" xmlns:outputxsl="dummy-uri">

  <xsl:namespace-alias stylesheet-prefix="outputxsl"
    result-prefix="xsl"/>

  <xsl:template match="/" xml:space="preserve">
    <outputxsl:stylesheet version="1.0">

      <outputxsl:template match="inputfield[@type='date']">
        <input size="2" name="day">
          <outputxsl:attribute name="value">
            <outputxsl:value-of select="day/text()"/>
          </outputxsl:attribute>
        </input>
        <select name="month">
          <xsl:for-each select="//months/month">
            <option>
              <outputxsl:attribute name="value">
                <xsl:value-of select="@number"/>
              </outputxsl:attribute>
              <outputxsl:if xml:space="default">
                <xsl:attribute name="test">month/text() =
                  <xsl:value-of select="@number"/>
                </xsl:attribute>
                <outputxsl:attribute name="SELECTED"/>
              </outputxsl:if>
              <xsl:value-of select="text()"/>
            </option>
          </xsl:for-each>
        </select>
        <input size="4" name="year">
          <outputxsl:attribute name="value">
            <outputxsl:value-of select="year/text()"/>
          </outputxsl:attribute>
        </input>
      </outputxsl:template>

    </outputxsl:stylesheet>
```

Result code:

```
</xsl:template>

</xsl:stylesheet>

<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:template match="inputfield[@type='date']">
      <input size="2" name="day">
        <xsl:attribute name="value">
          <xsl:value-of select="day/text()"/>
        </xsl:attribute>
      </input>
      <select name="month">

        <option>
          <xsl:attribute name="value">
            1
          </xsl:attribute>
          <xsl:if xml:space="default" test="month/text() = 1">
            <xsl:attribute name="SELECTED"/>
          </xsl:if>
          enero
        </option>

        <option>
          <xsl:attribute name="value">
            2
          </xsl:attribute>
          <xsl:if xml:space="default" test="month/text() = 2">
            <xsl:attribute name="SELECTED"/>
          </xsl:if>
          febrero
        </option>

        <option>
          <xsl:attribute name="value">
            3
          </xsl:attribute>
          <xsl:if xml:space="default" test="month/text() = 3">
            <xsl:attribute name="SELECTED"/>
          </xsl:if>
          marzo
        </option>

        <option>
          <xsl:attribute name="value">
            4
          </xsl:attribute>
          <xsl:if xml:space="default" test="month/text() = 4">
            <xsl:attribute name="SELECTED"/>
          </xsl:if>
          abril
        </option>
      </select>
    </template>
  </xsl:stylesheet>

```

```
</option>

<option>
  <xsl:attribute name="value">
    5
  </xsl:attribute>
  <xsl:if xml:space="default" test="month/text() = 5">
    <xsl:attribute name="SELECTED"/>
  </xsl:if>
  mayo
</option>

<option>
  <xsl:attribute name="value">
    6
  </xsl:attribute>
  <xsl:if xml:space="default" test="month/text() = 6">
    <xsl:attribute name="SELECTED"/>
  </xsl:if>
  junio
</option>

<option>
  <xsl:attribute name="value">
    7
  </xsl:attribute>
  <xsl:if xml:space="default" test="month/text() = 7">
    <xsl:attribute name="SELECTED"/>
  </xsl:if>
  julio
</option>

<option>
  <xsl:attribute name="value">
    8
  </xsl:attribute>
  <xsl:if xml:space="default" test="month/text() = 8">
    <xsl:attribute name="SELECTED"/>
  </xsl:if>
  agosto
</option>

<option>
  <xsl:attribute name="value">
    9
  </xsl:attribute>
  <xsl:if xml:space="default" test="month/text() = 9">
    <xsl:attribute name="SELECTED"/>
  </xsl:if>
  setembre
</option>

<option>
  <xsl:attribute name="value">
    10
```

```
</xsl:attribute>
<xsl:if xml:space="default" test="month/text() = 10">
<xsl:attribute name="SELECTED"/>
</xsl:if>
    octubre
</option>

<option>
    <xsl:attribute name="value">
        11
    </xsl:attribute>
    <xsl:if xml:space="default" test="month/text() = 11">
    <xsl:attribute name="SELECTED"/>
    </xsl:if>
    noviembre
</option>

<option>
    <xsl:attribute name="value">
        12
    </xsl:attribute>
    <xsl:if xml:space="default" test="month/text() = 12">
    <xsl:attribute name="SELECTED"/>
    </xsl:if>
    diciembre
</option>

</select>
<input size="4" name="year">
    <xsl:attribute name="value">
        <xsl:value-of select="year/text()"/>
    </xsl:attribute>
</input>
</xsl:template>

</xsl:stylesheet>
```

Example: Numbering paragraphs and chapters

The `xsl:number` element is a very powerful and versatile tool for inserting chapter (paragraph, footnote, list item, ...) numbers automatically. It allows both for the specification of the format of a number (numerals, letters, roman numerals, Hebrew letter-numbers) as for the calculation of the appropriate value for the current node.

The `value` attribute can be used to specify the numeric value to send to the output. In this example, we don't use the `value` attribute. The numeric value is calculated using the `level`, `count` and `from` attributes.

The output format of the calculated value is specified by the attributes `format`, `lang`, `letter-value`, `grouping-separator` and `grouping-size`. Note how one `xsl:number` element can be used to create paragraph numbers such as III.a.

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:apply-templates](#) [xsl:number](#) [xsl:value-of](#)

XML source:

```
<?xml version="1.0"?>
```

```
<book>
  <chapter title="It begins">
    <para title="First paragraph"/>
    <para title="2nd paragraph"/>
    <para title="3rd paragraph"/>
    <para title="4th paragraph"/>
  </chapter>
  <chapter title="The sequel">
    <para title="Paragraph"/>
    <para title="Yet another paragraph"/>
  </chapter>
  <chapter title="Final chapter">
    <para title="Da paragraph"/>
    <para title="Last paragraph"/>
  </chapter>
</book>
```

XSLT code:

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

  <xsl:template match="/">
    <formatted-book>
      <xsl:apply-templates select="book/chapter"/>
    </formatted-book>
  </xsl:template>

  <xsl:template match="chapter">
    Chapter <xsl:number format="I"/>:
    <xsl:value-of select="@title"/>
    <xsl:apply-templates select="para"/>
  </xsl:template>

  <xsl:template match="para">
    Paragraph <xsl:number level="multiple"
      count="para|chapter" format="I.a"/>:
    <xsl:value-of select="@title"/>
    (is paragraph nr. <xsl:number
      level="any" count="para" format="1"/>)
  </xsl:template>

</xsl:stylesheet>
```

Result code:

```
<?xml version="1.0" encoding="utf-8"?><formatted-book>
  Chapter I: It begins
  Paragraph I.a:
  First paragraph
  (is paragraph nr. 1)

  Paragraph I.b:
  2nd paragraph
  (is paragraph nr. 2)
```

```
Paragraph I.c:  
3rd paragraph  
(is paragraph nr. 3)  
  
Paragraph I.d:  
4th paragraph  
(is paragraph nr. 4)  
  
Chapter II: The sequel  
Paragraph II.a:  
Paragraph  
(is paragraph nr. 5)  
  
Paragraph II.b:  
Yet another paragraph  
(is paragraph nr. 6)  
  
Chapter III: Final chapter  
Paragraph III.a:  
Da paragraph  
(is paragraph nr. 7)  
  
Paragraph III.b:  
Last paragraph  
(is paragraph nr. 8)  
</formatted-book>
```

Example: Using different axes

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:apply-templates](#)
[xsl:call-template](#) [xsl:value-of](#)

XSLT and XPath functions used: [count](#)

XML source:

```
<?xml version="1.0"?>  
<book>  
  <chapter title="The sequel"  
    xmlns:annotation="http://some.annotation.com/namespace/">  
    <annotation:para title="Paragraph"/>  
    <para title="Yet another paragraph"  
      annotation:referral="Tom Sawyer"/>  
    <para title="Last paragraph"/>  
  </chapter>  
</book>
```

XSLT code:

```
<xsl:stylesheet version="1.0"
```

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <Overview>Overview of numbers of nodes on all axes
    <xsl:apply-templates select="@*|node()"/>
  </Overview>
</xsl:template>

<xsl:template match="@*|node()">
  <xsl:call-template name="axes-summary"/>
  <xsl:apply-templates select="@*|node()"/>
</xsl:template>

<xsl:template name="axes-summary">

  name(): <xsl:value-of select="name()"/>
  local-name(): <xsl:value-of
    select="local-name()"/>
  namespace-uri(): <xsl:value-of
    select="namespace-uri()"/>
  Axis self: <xsl:value-of
    select="count(self::*)/>
  Axis child: <xsl:value-of
    select="count(child::*)/>
  Axis descendant: <xsl:value-of
    select="count(descendant::*)/>
  Axis ancestor: <xsl:value-of
    select="count(ancestor::*)/>
  Axis preceding-sibling: <xsl:value-of
    select="count(preceding-sibling::*)/>
  Axis following-sibling: <xsl:value-of
    select="count(following-sibling::*)/>
  Axis attribute: <xsl:value-of
    select="count(attribute::*)/>
  Axis preceding: <xsl:value-of
    select="count(preceding::*)/>
  Axis following: <xsl:value-of
    select="count(following::*)/>
  Axis namespace: <xsl:value-of
    select="count(namespace::*)/>
</xsl:template>

</xsl:stylesheet>
```

Result code:

```
<?xml version="1.0" encoding="utf-8"?><Overview>Overview of
  numbers of nodes on all axes

  name(): book
  local-name(): book
  namespace-uri():
    Axis self: 1
    Axis child: 1
    Axis descendant: 4
```

```
Axis ancestor: 0
Axis preceding-sibling: 0
Axis following-sibling: 0
Axis attribute: 0
Axis preceding: 0
Axis following: 0
Axis namespace: 1

name():
local-name():
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 1
  Axis preceding-sibling: 0
  Axis following-sibling: 1
  Axis attribute: 0
  Axis preceding: 0
  Axis following: 4
  Axis namespace: 0

name(): chapter
local-name(): chapter
namespace-uri():
  Axis self: 1
  Axis child: 3
  Axis descendant: 3
  Axis ancestor: 1
  Axis preceding-sibling: 0
  Axis following-sibling: 0
  Axis attribute: 1
  Axis preceding: 0
  Axis following: 0
  Axis namespace: 2

name(): title
local-name(): title
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 2
  Axis preceding-sibling: 0
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 0
  Axis following: 0
  Axis namespace: 0

name():
local-name():
namespace-uri():
  Axis self: 0
  Axis child: 0
```

```
Axis descendant: 0
Axis ancestor: 2
Axis preceding-sibling: 0
Axis following-sibling: 3
Axis attribute: 0
Axis preceding: 0
Axis following: 3
Axis namespace: 0

name(): annotation:para
local-name(): para
namespace-uri(): http://some.annotation.com/namespace/
  Axis self: 1
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 2
  Axis preceding-sibling: 0
  Axis following-sibling: 2
  Axis attribute: 1
  Axis preceding: 0
  Axis following: 2
  Axis namespace: 2

name(): title
local-name(): title
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 3
  Axis preceding-sibling: 0
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 0
  Axis following: 2
  Axis namespace: 0

name():
local-name():
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 2
  Axis preceding-sibling: 1
  Axis following-sibling: 2
  Axis attribute: 0
  Axis preceding: 1
  Axis following: 2
  Axis namespace: 0

name(): para
local-name(): para
namespace-uri():
  Axis self: 1
```

```
Axis child: 0
Axis descendant: 0
Axis ancestor: 2
Axis preceding-sibling: 1
Axis following-sibling: 1
Axis attribute: 2
Axis preceding: 1
Axis following: 1
Axis namespace: 2

name(): title
local-name(): title
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 3
  Axis preceding-sibling: 0
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 0
  Axis following: 1
  Axis namespace: 0

name(): annotation:referral
local-name(): referral
namespace-uri(): http://some.annotation.com/namespace/
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 3
  Axis preceding-sibling: 0
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 0
  Axis following: 1
  Axis namespace: 0

name():
local-name():
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 2
  Axis preceding-sibling: 2
  Axis following-sibling: 1
  Axis attribute: 0
  Axis preceding: 2
  Axis following: 1
  Axis namespace: 0

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

```
Axis self: 1
Axis child: 0
Axis descendant: 0
Axis ancestor: 2
Axis preceding-sibling: 2
Axis following-sibling: 0
Axis attribute: 1
Axis preceding: 2
Axis following: 0
Axis namespace: 2

name(): title
local-name(): title
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 3
  Axis preceding-sibling: 0
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 0
  Axis following: 0
  Axis namespace: 0

name():
local-name():
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 2
  Axis preceding-sibling: 3
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 3
  Axis following: 0
  Axis namespace: 0

name():
local-name():
namespace-uri():
  Axis self: 0
  Axis child: 0
  Axis descendant: 0
  Axis ancestor: 1
  Axis preceding-sibling: 1
  Axis following-sibling: 0
  Axis attribute: 0
  Axis preceding: 4
  Axis following: 0
  Axis namespace: 0</Overview>
```

Example: Whitespace preserving and stripping

Whitespace stripping is one of the more confusing subjects in XSLT programming. What is important to understand is at which moments whitespace can be inserted or removed. The process of creating a result document from a source document and a stylesheet document has three of these moments:

1. Creating a tree structure in memory from the source document (some whitespace can get stripped)
2. Creating a tree structure in memory from the stylesheet document (normally most whitespace is stripped)
3. Creating a result document from the resulting tree structure in memory (some whitespace can be inserted)

The only whitespace that gets stripped are text nodes existing entirely of whitespace characters (space, tab, newline). If one non-whitespace character is included in the textnode, it will never be stripped.

In step 1, by default no whitespace is stripped. In step 2, all whitespace is stripped by default, except from `xsl:text` elements. In step 3, by default no whitespace is added.

To have whitespace stripped from the source document, you can place an `xml:space="preserve"` attribute in the source document. It will cause the descending whitespace nodes to be stripped. In the stylesheet, the `xsl:strip-space` and `xsl:preserve-space` top-level elements can be used to have certain whitespace nodes from the source document stripped or preserved.

The insertion of whitespace into the XML output document after building the result tree only happens when the `xsl:output` top-level element has its `indent` attribute set to `yes`. There is no way to specify whitespace insertion more exactly.

XSLT elements used: [xsl:stylesheet](#) [xsl:template](#) [xsl:apply-templates](#) [xsl:preserve-space](#) [xsl:strip-space](#) [xsl:value-of](#)

XML source:

```
<?xml version="1.0"?>
  <book>
    <chapter >
      content
      <para />
    </chapter>
    <chapter >
      <para />
      <para ></para>
      <para > </para>
      <para > content </para>
    </chapter>
  </book>
```

XSLT code:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:strip-space elements="*" />
<xsl:preserve-space elements="para" />
<xsl:template match="/">
  <root>
    <xsl:apply-templates select="book/chapter" />
  </root>
</xsl:template>

<xsl:template match="chapter">chapter:[<xsl:apply-templates
  select="para|text()" />]</xsl:template>
```

```
<xsl:template match="text()">[<xsl:value-of select="."/>]</xsl:template>

<xsl:template match="para">para:[<xsl:apply-templates
  select="text()" />]</xsl:template>

</xsl:stylesheet>
```

Result code:

```
<?xml version="1.0" encoding="utf-8"?><root>chapter:[[
  content
]para:[]]chapter:[para:[]para:[]para:[[]]para:[[] content []]]</root>
```

Compatibility overview

Axes

	w3c1	msxml2	msxml26	msxml3	msxml4	msxml.NET
child	X	X	X	X	X	X
parent	X	X	X	X	X	X
descendant	X	X	X	X	X	X
ancestor	X		X	X	X	X
attribute	X		X	X	X	X
ancestor-or-self	X		X	X	X	X
descendant-or-self	X		X	X	X	X
self	X	X	X	X	X	X
preceding-sibling	X			X	X	X
following-sibling	X			X	X	X
namespace	X			X	X	X
following	X			X	X	X
preceding	X			X	X	X

XPath Functions

	w3c1	msxml2	msxml26	msxml3	msxml4	msxml.NET
<u>boolean</u>	X		X	X	X	X
<u>ceiling</u>	X			X	X	X
<u>concat</u>	X		X	X	X	X
<u>contains</u>	X		X	X	X	X
<u>count</u>	X		X	X	X	X
<u>false</u>	X		X	X	X	X
<u>floor</u>	X			X	X	X
<u>id</u>	X	X	X	X	X	X
<u>lang</u>	X			X	X	X
<u>last</u>	X	X	X	X	X	X
<u>local-name</u>	X		X	X	X	X
<u>name</u>	X		X	X	X	X
<u>namespace-uri</u>	X		X	X	X	X
<u>normalize-space</u>	X		X	X	X	X
<u>not</u>	X		X	X	X	X
<u>number</u>	X		X	X	X	X
<u>position</u>	X		X	X	X	X

<u>round</u>	X			X	X	X	
<u>starts-with</u>	X		X	X	X	X	
<u>string</u>	X		X	X	X	X	
<u>string-length</u>	X		X	X	X	X	
<u>substring</u>	X		X	X	X	X	
<u>substring-after</u>	X		X	X	X	X	
<u>substring-before</u>	X		X	X	X	X	
<u>sum</u>	X			X	X	X	
<u>translate</u>	X		X	X	X	X	
<u>true</u>	X		X	X	X	X	
		w3c1	msxml2	msxml26	msxml3	msxml4	msxml.NET
<u>xsl:apply-imports</u>	X				X	X	X
<u>xsl:apply-templates</u>	X	X	X	X	X	X	X
<u>xsl:attribute</u>	X	X	X	X	X	X	X
<u>xsl:attribute-set</u>	X			X	X	X	X
<u>xsl:call-template</u>	X			X	X	X	X
<u>xsl:choose</u>	X	X	X	X	X	X	X
<u>xsl:comment</u>	X	X	X	X	X	X	X
<u>xsl:copy</u>	X	X	X	X	X	X	X
<u>xsl:copy-of</u>	X		X	X	X	X	X
<u>xsl:decimal-format</u>	X		X	X	X	X	X
<u>xsl:element</u>	X	X	X	X	X	X	X
<u>xsl:fallback</u>	X			X	X	X	X
<u>xsl:for-each</u>	X	X	X	X	X	X	X
<u>xsl:if</u>	X	X	X	X	X	X	X
<u>xsl:import</u>	X			X	X	X	X
<u>xsl:include</u>	X		X	X	X	X	X
<u>xsl:key</u>	X			X	X	X	X
<u>xsl:message</u>	X			X	X	X	X
<u>xsl:namespace-alias</u>	X			X	X	X	X
<u>xsl:number</u>	X			X	X	X	X
<u>xsl:otherwise</u>	X	X	X	X	X	X	X
<u>xsl:output</u>	X		X	X	X	X	X
<u>xsl:param</u>	X		X	X	X	X	X
<u>xsl:preserve-space</u>	X			X	X	X	X
<u>xsl:processing-instruction</u>	X	X	X	X	X	X	X
<u>xsl:sort</u>	X		X	X	X	X	X
<u>xsl:strip-space</u>	X		X	X	X	X	X

<u>xsl:stylesheet</u>	X	X	X	X	X	X
<u>xsl:template</u>	X	X	X	X	X	X
<u>xsl:text</u>	X	X	X	X	X	X
<u>xsl:transform</u>	X			X	X	X
<u>xsl:value-of</u>	X	X	X	X	X	X
<u>xsl:variable</u>	X		X	X	X	X
<u>xsl:when</u>	X	X	X	X	X	X
<u>xsl:with-param</u>	X		X	X	X	X

Functions

	w3c1	msxml2	msxml26	msxml3	msxml4	msxml.NET
<u>current</u>	X		X	X	X	X
<u>document</u>	X			X	X	X
<u>element-available</u>	X			X	X	X
<u>format-number</u>	X			X	X	X
<u>function-available</u>	X			X	X	X
<u>generate-id</u>	X			X	X	X
<u>key</u>	X			X	X	X
<u>system-property</u>	X			X	X	X
<u>unparsed-entity-url</u>	X			X	X	X

Implementations covered

w3c1

W3C 1.0 specification (recommendation)

[Go there...](#)

msxml2

MSXML 2.0 (IE5)

msxml26

MSXML 2.6 (January 2000 preview)

msxml3

MSXML 3.0

[Go there...](#)

msxml4

MSXML 4.0

[Go there...](#)

msxml.NET

XML classes in .NET Framework class library

[Go there...](#)