

<http://www.sqlservercentral.com/articles/Basics/20010422115709/78/>

Printed 2008/01/04 02:17PM

XML IN 20 MINUTES!

By [Leon Platt](#), 2001/04/22

XML IN 20 MINUTES!

This article will quickly dive head first right into XML. I will take an everyday object and attempt to describe it using XML. Then I will load the XML file into a XML document object model (DOM). Next, I will demonstrate how to query the XML document using XPath and perform some minor manipulation of the DOM.

All of this functionality will be preformed using a simple Visual basic application and the Microsoft parser version 3.0. The final objective of this article will be to produce an ActiveX control which will query the pubs SQL Server database and return a list of book titles in XML format.

LETS GET STARTED !

If any of you have ever picked up a book and tried to learn XML on your own you were probably confronted with the same confusing bombardment of information that I was. DTDs, XML Schema, name spaces, XPath, XPointers, XSL , XSLT, DOMs, SAX, SOAP, YIKES I GIVE UP. To make matters worse, most of this material is based on proposed implementations and much of the sample software is buggy and unpredictable. There must be a million ways to implement and use XML, this can all be very complicated. But guess what; XML can be simple too. If we ignore DTDs, XML Schemas, namespaces, etc.

In my quest to get you up and running with XML quickly, I am going to ignore a good portion of the information you will find in many books on the subject. The first items I will ignore are namespaces and schemas. You may think this is strange since most books start with these subjects, but think of XML as a tool to get a job done, just like a hammer is a tool. To use a hammer, do I need to understand how to build a house? What if all I want to do is hang a picture? XML is the same way, it can be really complex and full of confusing specifications making it versatile enough to use in hundreds if not thousands of applications, but it can also be very simple if some things are ignored. In this article, I will concentrate on the XML necessary to solve a specific problem.

So what exactly is the problem? Well lets suppose I want to describe a simple object such as a cup using XML. Why would I want to use XML to do this? Well, first of all, that's what XML does. XML describes data. In my example; the cup is the data. In real life, the data could be a word document, a spreadsheet, an image, a book, a record from a database, or maybe even a VB or C++ class. Secondly, XML is extensible. XML lets me create as many tags as I want to describe our data and the tags can be defined however I want, as long as I follow some simple rules. Lastly, because it is quickly becoming a universal standard. If there is life on mars, you can bet they will understand my XML file.

What are some of the properties that allow me to describe a cup?

```
The material its made of
  If the material is transparent or not
the height in inches
the number of ounces it will hold
its contents
  description of any solid objects and the quantity
  description of any liquid substance and the volume
  description of any other substance and the quantity
whether or not it has a lid
```

So what would this look like in XML?

```
<?xml version="1.0"?>
<CUP>
<MATERIAL transparent="yes">glass</MATERIAL>
<HEIGHT units="inches">6</HEIGHT>
<VOLUME units="ounces">16</VOLUME>
<CONTENTS>
  <SOLID qty="2">ice cube</SOLID>
  <SOLID qty="1">straw</SOLID>
  <LIQUID qty="3" units="ounces">water</LIQUID>
  <OTHER qty="0"/>
</CONTENTS>
<LID>yes</LID>
</CUP>
```

Note, the first line of the XML file (`<?xml version="1.0"?>`) is known as a processing command. For now, just know that it needs to be there. The neat thing about the XML file, is that anyone can get a very good idea of what its purpose is just by looking at it. Understand that this is not the only valid XML representation for a cup. If I asked ten people to create a XML file for a cup; given the same properties, they may all very well each produce something different but correct. This could be a problem. Maybe not for us humans, but if a computer is reading the file it would probably be a good idea to give it a file that it knows something about. That's where namespaces and schemas come in to the equation. To put things simple, schemas are used to specify a valid structure for a XML file.

This would be a good time to talk about a few simple XML rules that need to be followed:

XML RULE #1: a valid XML file will conform exactly to its specified schema. However, to make things simple, none of my examples will use schemas. Thus, none of my XML files will be "valid" as far as a parser is concerned. But, guess what, I don't care. I'm not building a house, I'm only hanging a picture. I'll explain more about this later when I get into using the XML DOM.

XML RULE #2: If you are a VB programmer; XML is case sensitive. XML is case sensitive. XML is case sensitive. Write this sentence 1000 times and don't forget it.

XML RULE #3: Tag names are called Elements, and every beginning tag must have an ending tag. This is known as well-formed XML. This is very important because a XML file will not parse and will not load into the DOM unless it is well-formed. Note: if elements do not contain a value and do not contain other elements then the short hand version of the end tag `<Element />` may be used instead of the long version `<Element></Element>`. This can be seen in my example above for the element `<OTHER qty="0"/>` in the contents section of the cup XML file.

XML RULE #4: Elements can contain attributes and the values of the attributes must be contained in quotes (either single or double).

XML RULE #5: It is ok to re-use attribute names, but make sure your element names are unique throughout the entire file. In the example above, the attribute `qty` has a different meaning depending on whether it is used in the element `<SOLID>`, `<LIQUID>`, or `<OTHER>`. The meaning of an attribute depends on the context in which it is being used. Whereas, the meaning of an element always means the same thing no matter where it is found in the file. In our example above the elements `<SOLID>` and `<HEIGHT>` always have the same meaning in our document. `<SOLID>` is always used to describe a solid that is present as part of the cups contents, and `<HEIGHT>` always describes the height of the cup.

XML RULE #6: There are some special characters in xml that can not be used directly as data because they would interfere with the syntax of tags and attributes. Thus, these characters must be escaped out using the `&` character and a special code. (`&` must be replaced with `&`;) (`"` must be replaced with `"`;) (`<` must be replaced with `<`;) (`>` must be replaced with `>`;) and (`'` must be replaced with `'`;) To get around this a special xml processing instruction `<![CDATA[...]]>` can be used, where the `"...."` portion can be any character string that does not include the `"]]>"` string literal. CDATA sections are a method of including data that contains characters which would otherwise be recognized as markup. CDATA sections may occur anywhere that character data may occur, but they cannot be nested.

Ok, so what is the XML DOM & why is it needed?

The XML DOM allows programmers to load into memory a representation of the entire XML file. Once the XML file is loaded into memory, it can then be manipulated using the properties, methods and events of the DOM. This is where the usefulness of XML really shines. The DOM makes it very easy to retrieve and manipulate information in the XML file. I am not going to go over everything that can be done with the DOM, but I will cover some basic items which will help accomplish the goals of this article. I will take the XML file for a cup that was just created, load the XML file into a DOM so that it can be examined, then do some minor manipulation of the data by querying the DOM. I will save the major manipulation methods of the DOM for my next article dealing with client-side XML. Note: although the DOM is great for programmers it does have the drawback of consuming massive amounts of memory and system resources. It is precisely this reason that you will see references to another method of parsing XML files known as SAX. My articles will not get into the concepts behind SAX but there is plenty of information available in the XML SDK for those who wish to become experts.

XML DOM – LOADING THE XML FILE

Lets look at an example using Microsoft's XML parser version 3.0 (`msxml3.dll`) to see how this works. If you do not have version 3.0 of the parser it can be [downloaded](#) from Microsoft. . The msxml 3.0 release installs the Microsoft XML Parser in side-by-side mode, which means that when you download and install msxml 3.0, it does not replace any previously installed version of the parser, You can also run msxml 3.0 in Replace mode, using [xmlinst.exe](#), which means that application references to previous versions of msxml are remapped to point to the new msxml 3.0 dll. **Note, however, that running msxml 3.0 in Replace mode can leave your computer in an unstable state.** For more information about running msxml 3.0 in Replace mode, see the Knowledge Base article [Application Errors Occur After You Run Xmlinst.exe on Production Servers](#). Suppose I have saved the sample XML cup file as `"http://web_server/xml/cup.xml"` (local path `C:\inetpub\wwwroot\xml\cup.xml`) and now I want to load this file up into a XML DOM. The code below assumes the parser was loaded in default side-by-side mode and

uses version-dependent progIDs to specifically specify version 3.0 of the parser. [Click here](#) for a further explanation of side-by-side mode and version-dependent progIDs.

Visual Basic 6.0 code:

(need to set a reference to Microsoft XML, v3.0)

```
Dim xmlDoc as MSXML2.DOMDocument30
Set xmlDoc = New DOMDocument30
xmlDoc.async = False
xmlDoc.validateOnParse = False
xmlDoc.load ("c:\inetpub\wwwroot\xml\cup.xml")
msgBox xmlDoc.xml
```

ASP Server-Side VB script code:

```
Dim xmlDoc
Set xmlDoc = Server.CreateObject("Msxml2.DOMDocument.3.0")
xmlDoc.async = False
xmlDoc.validateOnParse = False
xmlDoc.load "/xml/cup.xml"
```

ASP Server-Side Java Script code:

```
var xmlDoc = Server.CreateObject("Msxml2.DOMDocument.3.0");
xmlDoc.async = false;
xmlDoc.validateOnParse = false;
xmlDoc.load ("/xml/cup.xml");
```

EXPLANATION OF THE CODE - Lets walk through the VB6 code

Line 1: Dim xmlDoc as MSXML2.DOMDocument30

This first statement requires a reference set to "Microsoft XML, v3.0". In this line, I am declaring the variable xmlDoc to be a XML document (DOM). MSXML2 is the library and this is always MSXML2 (you may see older code that uses the library msxml) (also do not be tempted to put MSXML3 for version 3, this has nothing to do with the parser version and it will not work). DOMDocument30 specifies a XML document object (DOM) conforming to version 3.0. You may also see code samples where the line may be dim xmlDoc as MSXML2.DOMDocument. Notice there is no 30 after DOMDocument, this is the version independent way to set up a XML document. The version independent method will use the currently registered parser, whatever version it may be. The problem with the independent method is that the default registered version of the parser will vary from machine to machine. If you want to use the independent method, you better make sure your code is generic enough to work with any version of the parser or it may break. You can not assume that every user will have the version parser you are expecting. The advantage to using the independent method is that when a new version of the parser comes out the application will support it automatically without recompiling.

Line 2: Set xmlDoc = new DOMDocument30

This line actually initializes the variable xmlDoc to a new XML document (DOM) version 3.0.

Line 3: xmlDoc.async = False

XML files can be loaded either synchronously or asynchronously. If xmlDoc.async=False then the entire XML file will be loaded before control is given back to the calling program. If xmlDoc.async=True then the load function will immediately return control to the calling program even though the XML file may not be entirely loaded.

Line 4: xmlDoc.validateOnParse = False

When loaded the parser can be instructed to validate the XML file against a schema (validateOnParse=True) or we can ignore schemas and validation entirely by setting (validateOnParse = False).

Line 5: xmlDoc.load ("C:\inetpub\wwwroot\xml\cup.xml")

This line calls the load method of the DOMDocument. There are two versions of the load method. The one I am calling here loads a **file** into the DOM and it must be passed a valid path and filename. The second version of the load method is used to load a **xml string** into a DOM and it must be passed a well-formed xml string. The string version of the load method would be called using xmlDoc.loadXML("well-formed xml string"). I will demonstrate this later.

Line 6: MsgBox xmlDoc.xml

This line will display the xml for the entire document. The results should be the same as the original xml file.

EXAMINING THE XML DOM

Create a new standard.exe project in visual basic. Insert the sample code into the forms load method. Make sure you create a reference to "Microsoft XML v3.0". To do this click Project-->References, then scroll through the list until you find it. Note: the version 3 of the parser must be installed or you will not find the reference in your list. Set a breakpoint on the last line of the

code. The line that reads `msgbox(xmlDoc.xml)`. Run the application in debug mode. When the breakpoint is hit and the application stops; bring up the locals window and examine the DOM. You can learn a lot about the DOM by walking through it in the locals window. The locals window should look similar to the figure on the next page. Some interesting properties of the DOM are:

The XML DOM **always** contains two top level nodes:

Item1 is the root element of the document (**ignore this node**)

Item2 is actually the first element of the document (**remember this**)

nodeName or **baseName** - this can be used to find the name of an Element or Attribute

nodeType - use this to retrieve the type of the current node.

nodeValue - use this to get the data value of the node

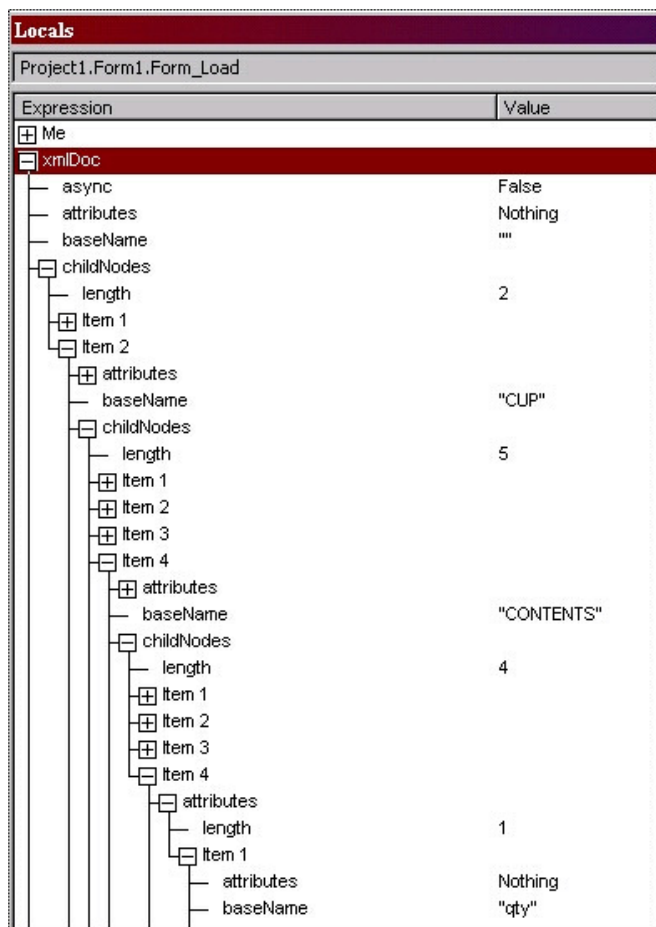
childNodes - is a collection of children nodes that exist in the tree at a level directly bellow the current node. Child nodes can be element nodes, text nodes or CDATA nodes. There are also other types of nodes that I will not discuss, but you can find a good description of them in the XML SDK.

attributes - is a collection of attribute nodes for the current element.

length - use this to find the number of nodes in the tree at a level directly bellow the current node.

xml - this property exists on all nodes and can be used to get the xml string representing the current position in the document. The xml string starts with the current node and traverses down the rest of the tree. This is a very useful property. Play around with it and see what you get.

EXAMINING THE XML DOM - A successful load



EXAMINING THE XML DOM - Element nodes

Element nodes can contain other element child nodes, attribute child nodes, text child nodes, or CDATA child nodes. From the figure bellow the following information can be obtained about the current node "SOLID" :

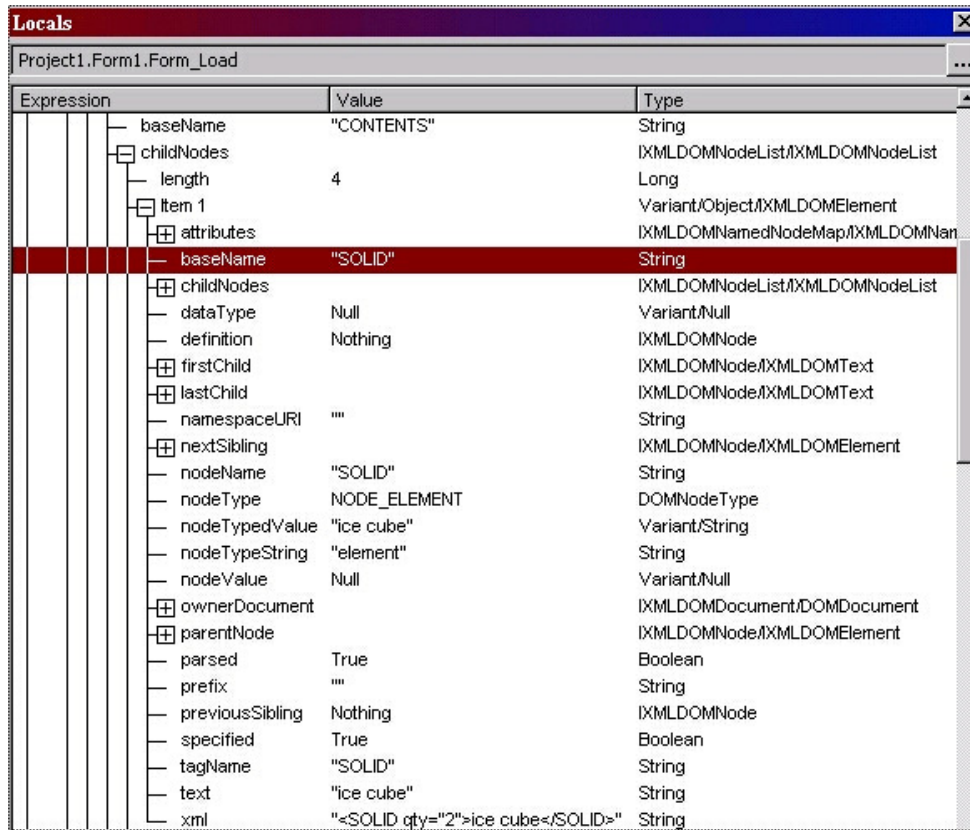
nodeType - The current nodes type = `NODE_ELEMENT` - the current node is an element.

nodeName or **baseName** or **tagName** - The current nodes (Elements) name is = `SOLID`

It's parent element is `CONTENTS` which has 4 children

You can't see it from the figure bellow but `SOLID` has one `childNodes`, which is of type text. (see text nodes on a following page)

text - "ice cube" this is a short cut method of obtaining the current nodes value without actually navigating to the child text node.



Expression	Value	Type
baseName	"CONTENTS"	String
childNodes	IXMLDOMNodeList/IXMLDOMNodeList	IXMLDOMNodeList/IXMLDOMNodeList
length	4	Long
Item 1	Variant/Object/IXMLDOMElement	Variant/Object/IXMLDOMElement
attributes	IXMLDOMNamedNodeMap/IXMLDOMNamedNodeMap	IXMLDOMNamedNodeMap/IXMLDOMNamedNodeMap
baseName	"SOLID"	String
childNodes	IXMLDOMNodeList/IXMLDOMNodeList	IXMLDOMNodeList/IXMLDOMNodeList
dataTypes	Null	Variant/Null
definition	Nothing	IXMLDOMNode
firstChild	IXMLDOMNode/IXMLDOMText	IXMLDOMNode/IXMLDOMText
lastChild	IXMLDOMNode/IXMLDOMText	IXMLDOMNode/IXMLDOMText
namespaceURI	""	String
nextSibling	IXMLDOMNode/IXMLDOMElement	IXMLDOMNode/IXMLDOMElement
nodeName	"SOLID"	String
nodeType	NODE_ELEMENT	DOMNodeType
nodeValue	"ice cube"	Variant/String
nodeTypeString	"element"	String
nodeValue	Null	Variant/Null
ownerDocument	IXMLDOMDocument/DOMDocument	IXMLDOMDocument/DOMDocument
parentNode	IXMLDOMNode/IXMLDOMElement	IXMLDOMNode/IXMLDOMElement
parsed	True	Boolean
prefix	""	String
previousSibling	Nothing	IXMLDOMNode
specified	True	Boolean
tagName	"SOLID"	String
text	"ice cube"	String
xml	"<SOLID qty=2>ice cube</SOLID>"	String

EXAMINING THE XML DOM - Attribute nodes

Attribute nodes can contain only text child nodes or CDATA child nodes. From the figure bellow the following information can be obtained about the current node "qty" :

nodeType - The current nodes type = `NODE_ATTRIBUTE` - the current node is an attribute.

nodeName or **baseName** - The current nodes (Attributes) name is = `qty`

You can't see it from the figure bellow, but `qty` has one `childNodes`, which is of type text. (see text nodes on a following page)

text or **value** - "2" this is a short cut method of obtaining the current nodes value without actually navigating to the child text node.

Locals	
Project1.Form1.Form_Load	
Expression	Value
baseName	"CONTENTS"
childNodes	
length	4
Item 1	
attributes	
length	1
Item 1	
attributes	Nothing
baseName	"qty"
childNodes	
data Type	Null
definition	Nothing
firstChild	
lastChild	
name	"qty"
namespaceURI	""
nextSibling	Nothing
nodeName	"qty"
nodeType	NODE_ATTRIBUTE
nodeTypedValue	"2"
nodeTypeString	"attribute"
nodeValue	"2"
ownerDocument	
parentNode	Nothing
parsed	True
prefix	""
previousSibling	Nothing
specified	True
text	"2"
value	"2"
xml	"qty="2"

EXAMINING THE XML DOM - Text nodes and CDATA nodes

Text and CDATA nodes do not contain child nodes. Text nodes contain parsed text data for its parent node. CDATA nodes contain unparsed text data for its parent node. CDATA nodes are created when the CDATA processing instruction is wrapped around data in the xml file. This is useful for cases when the data contains special characters. The CDATA processing tag tells the parser not to parse the data and to accept all characters within the tag as data. CDATA sections are especially useful when trying to embed code in an xml file. From the figure bellow the following information can be obtained about the current text node.

NodeType - The current nodes type = NODE_TEXT - the current node is an text data.

nodeName - The current nodes (text) name is = #text - **all text nodes are named #text**

data or text or value - "2" - this is the current nodes data

Locals	
Project1.Form1.Form_Load	
Expression	Value
baseName	"CONTENTS"
childNodes	
length	4
Item 1	
attributes	
length	1
Item 1	
attributes	Nothing
baseName	"qty"
childNodes	
length	1
Item 1	
attributes	Nothing
baseName	""
childNodes	
data	"2"
dataType	Null
definition	Nothing
firstChild	Nothing
lastChild	Nothing
length	1
namespaceURI	""
nextSibling	Nothing
nodeName	"#text"
nodeType	NODE_TEXT
nodeTypedValue	"2"
nodeTypeString	"text"
nodeValue	"2"
ownerDocument	
parentNode	

EXAMINING THE XML DOM - Errors loading the document

The **parseError** section of the DOM can be beneficial to tracking down problems with loading the xml document. If I remove the ending tag from OTHER in our sample file and run the program again I get the following results. The first piece of helpful information is that our **nextSibling** is Nothing. Also, if you looked at **childNodes** you would find the length equal to zero. Both these things indicate that the xml document did not load. To find out why it did not load I open up the **parseError** node to retrieve all the specific error information.

Locals	
Project1.Form1.Form_Load	
Expression	Value
xmlDoc	
async	False
attributes	Nothing
baseName	""
childNodes	
dataType	Null
definition	Nothing
doctype	Nothing
documentElement	Nothing
firstChild	Nothing
implementation	
lastChild	Nothing
namespaces	
namespaceURI	""
nextSibling	Nothing
nodeName	"#document"
nodeType	NODE_DOCUMENT
nodeTypedValue	Null
nodeTypeString	"document"
nodeValue	Null
ownerDocument	Nothing
parentNode	Nothing
parsed	True
parseError	
errorCode	-1072896659
filepos	288
line	11
linepos	3
reason	"End tag 'CONTENTS' does not match the start tag 'OTHER'. "
srcText	"</CONTENTS>"
url	"file:///c:/xml/cup.xml"

XML DOM – QUERYING THE XML DOCUMENT

Ok, so I have shown how to get an xml file into the DOM, but what do I do with it now? Well, one of the most common things you will want to do is query the xml document. To accomplish this you could traverse the document until you find what you are looking for, but most likely you will use one of two methods of the XmlDocument or current node (context node). The two methods used to query for nodes in our previous example would be `xmlDoc.SelectSingleNode(patternString)` to return one node and `xmlDoc.SelectNodes(patternString)` to return a list of nodes. The `patternString` parameter that is passed into these methods consists of a query. This `patternString` can be formatted in one of two ways. Either as a XSL query or as a XPath query. The XPath version is the newer and more preferred method of querying xml documents. The format of the `patternString` must be set before any queries are run using the two indicated methods, otherwise the default XSL format will be used. To set the format of the `patternString` you will use the `setProperty("SelectionLanguage", "format")`. To change the queries in our example to use a XPath `patternString`, I will issue the following command: `setProperty("SelectionLanguage","XPath")`. In my opinion, XPath is probably one of the most important XML technologies to learn. I will introduce some simple XPath queries, but I highly recommend doing some independent research in this area. A good starting place would be the Microsoft XML SDK. I have also come across some pretty good tutorials on various web sites. Another avenue of exploration would be to write a simple VB application to allow you to enter XPath queries and have the results displayed. You can probably find a freeware application to do this but XPath is still fairly new and may not be entirely supported by every application that you find.

USING XPATH TO QUERY THE DOM

Lets add some additional code to the end of our previous example to return us just the contents of our cup:

```
Dim objNode As IXMLDOMNode
Dim objListOfNodes As IXMLDOMNodeList
xmlDoc.setProperty "SelectionLanguage", "XPath"
MsgBox "Your cup contains the following items"
Set objListOfNodes = xmlDoc.selectNodes("//CONTENTS/*")
For Each objNode In objListOfNodes
    MsgBox objNode.Text
Next
```

Run the program and see what you get. Notice that you should get four message boxes telling indicating what is in the cup. The last message box should be blank because the Element "OTHER" does not contain text. Lets fix the query to return all the contents of the cup where the `qty>0`. Change the line of code that does the query to the following:

```
Set objListOfNodes = xmlDoc.selectNodes("//CONTENTS/*[@qty>0]")
```


COOL! Now lets add one more query to find out if our cup has a lid or not. Add the following code to the end of the previous code:

```
Set objNode = xmlDoc.SelectSingleNode("/CUP/LID")
if objNode.text="yes" then
    MsgBox "We have a lid"
else
    MsgBox "No lid on this cup"
end if
```

Lets go over the code line by line:

Line 1: Dim objNode As IXMLDOMNode

This line defines the variable objNode as a variable of type xml document node. It is important to realize that a XML document node is an object. It is not a value. It consists of itself as well as all of its attributes and childNodes. In this manner you can prune a entire branch off of a tree by selecting the correct node.

Line 2: Dim objListOfNodes As IXMLDOMNodeList

This line defines the variable objListOfNodes as a variable of type xml document node list (a group of nodes).

Line 3: xmlDoc.setProperty "SelectionLanguage", "XPath"

This line sets our patternString format to XPath.

Line 4: MsgBox "Your cup contains the following items:"

Line 5: Set objListOfNodes = xmlDoc.selectNodes("//CONTENTS/*[@qty>0]")

This line runs a XPath query that will return a group of nodes and store them in the variable objListOfNodes. The query breaks down like this:

//CONTENTS - get all the CONTENTS elements in the xml document. This retrieves one element node from our sample XML file (<CONTENTS>). Note: // is short for in the entire xml document.

/* - from the list of CONTENTS element nodes, get all (* - short for all) the child elements. This narrows our results down to four element nodes (<SOLID><SOLID><LIQUID><OTHER>). These four nodes fall directly under the CONTENTS node.

[@qty>0] - for each child element test its qty attribute (@ - short for attribute) to see if it is greater than 0. If it is keep the element node otherwise discard it. Anything inside of [] in an XPath query gets resolved to true or false. If the results are true then the node remains. If the results are false, then the node is discarded. This narrows the query results down to three element nodes (<SOLID><SOLID><LIQUID>).

Line 6-8: For Each objNode In objListOfNodes / MsgBox objNode.Text / Next

These lines will display the value of each element node that matched the query. ("ice cube" , "straw" , "water")

Line 9: Set objNode = xmlDoc.SelectSingleNode("/CUP/LID")

This line queries for all the LID elements that exist directly under CUP elements that exist directly under the ROOT (when a query starts with / it means start at the root). This works just like a directory path, thus the name XPath. In the sample XML file the query will return the LID element which contains a value of "yes". The important thing to note here is that I am forcing the query to begin at the ROOT of the XML document. Queries will not always start with the ROOT, normally queries will start at the current node (context node). In the example this is irrelevant because my context node (xmlDoc) is the ROOT of the XML document (**this is not always the case**).

Line 10-15: if objNode.text="yes" then / MsgBox "We have a lid" /
else / MsgBox "No lid on this cup" /end if

This line will display the message "We have a lid" because the text property of the LID element is "yes".

CONVERTING ADO TO XML

Now that you understand the basics of XML, lets create an activeX control that will convert an ADO record set to XML. My goal is to run a query against the titles table in the pubs database and return the titles in XML format. I will use the results from the activeX control in my next article "client-side xml". Simple you say, ADO has a method to save as XML, right? Well yes, but if I let ADO create the XML I will get a version of XML that is nasty to work with. ADO will produce a XML file with namespaces and I am trying to ignore these for now. Secondly, ADO will produce a XML file that is in attribute form. In other words each record is an element (z:row) and **each field is an attribute**:

```

<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
      xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
      xmlns:rs='urn:schemas-microsoft-com:rowset'
      xmlns:z='#RowsetSchema'>
<s:Schema id='RowsetSchema'>
  <s:ElementType name='row' content='eltOnly'>
    <s:AttributeType name='title_id' rs:number='1' rs:writeunknown='true'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='6' rs:maybenull='false'
    </s:AttributeType>
    <s:AttributeType name='title' rs:number='2' rs:writeunknown='true'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='80' rs:maybenull='false'
    </s:AttributeType>
    <s:AttributeType name='type' rs:number='3' rs:writeunknown='true'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='12' rs:fixedlength='true'
    </s:AttributeType>
    <s:AttributeType name='price' rs:number='4' rs:nullable='true' rs:writeunknown='true'>
      <s:datatype dt:type='number' rs:dbtype='currency' dt:maxLength='8' rs:precision='10'
    </s:AttributeType>
    <s:AttributeType name='ytd_sales' rs:number='5' rs:nullable='true' rs:writeunknown='true'>
      <s:datatype dt:type='int' dt:maxLength='4' rs:precision='10' rs:fixedlength='true'
    </s:AttributeType>
    <s:AttributeType name='notes' rs:number='6' rs:nullable='true' rs:writeunknown='true'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='200' />
    </s:AttributeType>
    <s:AttributeType name='pubdate' rs:number='7' rs:writeunknown='true'>
      <s:datatype dt:type='dateTime' rs:dbtype='timestamp' dt:maxLength='16' rs:scale='1'
      rs:maybenull='false' />
    </s:AttributeType>
    <s:extends type='rs:rowbase' />
  </s:ElementType>
</s:Schema>
<rs:data>
  <z:row title_id='BU1032' title='The Busy Executive's Database Guide' type='business' price='1'
    ytd_sales='4095' notes='An overview of available database systems with emphasis on commo
    pubdate='1991-06-12T00:00:00' />
  <z:row title_id='BU1111' title='Cooking with Computers: Surreptitious Balance Sheets' type='busin
    ytd_sales='3876' notes='Helpful hints on how to use your electronic resources to the bes
</rs:data>
</xml>

```

I would like to produce a XML file that is in element form, where each record will be wrapped in a <BOOK> tag and **each field will be an element** under the <BOOK> tag. The syntax of my xml string will be as follows:

```

<TITLES>
<BOOK><data from table
  <FIELD prettyname="Book identification number" tablename="titles" columnname="title_id" datatype="num
  <FIELD prettyname="Title of the book" tablename="titles" columnname="title" datatype="text" filter=""
  <FIELD prettyname="Type of book" tablename="titles" columnname="type" datatype="text" filter=""><data
  <FIELD prettyname="Price of the book" tablename="titles" columnname="price" datatype="number" filter=""
  <FIELD prettyname="Year to date sales" tablename="titles" columnname="ytd_sales" datatype="number" fi
  <FIELD prettyname="Date published" tablename="titles" columnname="pubdate" datatype="date" filter=""
</BOOK>
</TITLES>

```

By the way, what I have just accomplished is to create a schema for my xml string. Now if I wanted to validate the structure of XML against the schema, all I would need to do is convert the schema into a valid format. Either DTD or XDR syntax. Notice that I have added several attributes to each <FIELD> element. One reason for this is that this information may be useful on the client. The prettyname can be used as labels for the data. The datatype attribute could be used for client side data validation. But to be honest, the real reason these attributes exist is that they have a very special purpose in a XSL template file that I regularly use for building "SQL where clauses". Maybe I will publish a short article demonstrating this soon. The template is very usefull and pretty cool. When the xml structure is applied to the actual data from the titles table the results will look like the sample below:

```

<TITLES>
  <BOOK>The Busy Executive's Database Guide
    <FIELD prettyname="Title Identification Number" tablename="titles" gcolumnname="title_id" datat
    <FIELD prettyname="Title of the Book" tablename="titles" gcolumnname="title" datatype="text" gf
    <FIELD prettyname="Type of Book" tablename="titles" gcolumnname="type" datatype="text" gfilter=
    <FIELD prettyname="Price of the Book" tablename="titles" gcolumnname="price" datatype="number"
    <FIELD prettyname="Year to date sales" tablename="titles" gcolumnname="ytd_sales" datatype="num
    <FIELD prettyname="Notes about the book" tablename="titles" gcolumnname="notes" datatype="memo"
    <FIELD prettyname="Date Published" tablename="titles" gcolumnname="pubdate" datatype="date" gfi
  </BOOK>
  <BOOK>Cooking with Computers: Surreptitious Balance Sheets
    <FIELD prettyname="Title Identification Number" tablename="titles" gcolumnname="title_id" datat
    <FIELD prettyname="Title of the Book" tablename="titles" gcolumnname="title" datatype="text" gf
    <FIELD prettyname="Type of Book" tablename="titles" gcolumnname="type" datatype="text" gfilter=
    <FIELD prettyname="Price of the Book" tablename="titles" gcolumnname="price" datatype="number"
    <FIELD prettyname="Year to date sales" tablename="titles" gcolumnname="ytd_sales" datatype="num
    <FIELD prettyname="Notes about the book" tablename="titles" gcolumnname="notes" datatype="memo"
    <FIELD prettyname="Date Published" tablename="titles" gcolumnname="pubdate" datatype="date" gfi
  </BOOK>
</TITLES>

```

Now I have something that I can work with! The XML is very basic and straight forward. Please come back for Article 2 in this series where I will demonstrate how easy it is to use this XML on the client ...

Listing 1 - CUP.XML

```
<?xml version="1.0"?>
<CUP>
<MATERIAL transparent="yes">glass</MATERIAL>
<HEIGHT units="inches">6</HEIGHT>
<VOLUME units="ounces">16</VOLUME>
<CONTENTS>
    <SOLID qty="2">ice cube</SOLID>
    <SOLID qty="1">straw</SOLID>
    <LIQUID qty="3" units="ounces">water</LIQUID>
    <OTHER qty="0"/>
</CONTENTS>
<LID>yes</LID>
</CUP>
```

Listing 2 - Load Cup.xml into a DOM

```
Dim xmlDoc As MSXML2.DOMDocument30
Set xmlDoc = New DOMDocument30
xmlDoc.async = False
xmlDoc.validateOnParse = False
xmlDoc.Load ("c:\inetpub\wwwroot\xml\cup.xml")
MsgBox xmlDoc.xml
Dim objNode As IXMLDOMNode
Dim objListOfNodes As IXMLDOMNodeList
xmlDoc.setProperty "SelectionLanguage", "XPath"
MsgBox "Your cup contains the following items"
Set objListOfNodes = xmlDoc.selectNodes("//CONTENTS/*[@qty>0]")
For Each objNode In objListOfNodes
    MsgBox objNode.Text
Next
Set objNode = xmlDoc.selectSingleNode("/CUP/LID")
If objNode.Text = "yes" Then
    MsgBox "We have a lid"
Else
    MsgBox "No lid on this cup"
End If
```

Listing 3 - ActiveX Control: ADO to XML (WebClass.dll)(xmlControl.cls)

```
Option Explicit

'Declare Database variables
Private m_dbConnection As New ADODB.Connection
Private m_dbCommand As ADODB.Command
Private m_adoRs As ADODB.Recordset
Private m_adoErrors As ADODB.Errors
Private m_adoErr As Error
Public nCommandTimeout As Variant
Public nConnectionTimeout As Variant
Public strConnect As Variant
Public strAppName As String
Public strLogPath As String
Public strDatabase As String
Public strUser As String
Public strPassword As String
Public strServer As String
Public strVersion As String
Public lMSADO As Boolean
'Private Global Variables
Private gnErrNum As Variant
Private gstrErrDesc As Variant
Private gstrErrSrc As Variant
Private gstrDB As String
Private gstrADOError As String
Private Const adLeonNoRecordset As Integer = 129
Private gtableName(6) As String
Private gcolumnName(6) As String
Private gprettyName(6) As String
Private gdatatype(6) As String
Private gfilter(6) As String

Private Function OpenDatabase()

If Len(strConnect) = 0 Then 'set up defaults if they were not passed
    If Len(strDatabase) = 0 Then
        strDatabase = "pubs"
    End If

    If nConnectionTimeout = 0 Then
        nConnectionTimeout = 600
    End If
```

```

    If nCommandTimeOut = 0 Then
        nCommandTimeOut = 600
    End If

    If Len(strAppName) = 0 Then
        strAppName = "xmlControl"
    End If

    If Len(strUser) = 0 Then
        strUser = "sa"
    End If

    If Len(strPassword) = 0 Then
        strPassword = ""
    End If

    strConnect = "Provider=SQLOLEDB.1; " & _
        "Application Name=" & strAppName & _
        "; Data Source=" & strServer & "; Initial Catalog=" & strDatabase & "; " & _
        " User ID=" & strUser & "; Password=" & strPassword & ";"

End If

'connect to SQL Server and open the database
On Error GoTo SQLErr 'turn on error handler
With m_dbConnection
    .ConnectionTimeout = nConnectionTimeout
    .CommandTimeout = nCommandTimeOut
    .Open strConnect 'open the database using the connection string
End With
On Error GoTo 0 'turn off error handler

OpenDatabase = True 'database opened successfully

Exit Function

SQLErr:
Call logerror("OPEN")
OpenDatabase = False

End Function

Private Function BuildSQLWhere(tmpWhere) As String

    'This is for the future

End Function

Public Function GetTitlesXML(Optional xmlWhere As Variant) As String

Dim whereClause As String
Dim strSQL As String

Call OpenDatabase 'open the pubs database

If IsMissing(xmlWhere) Then 'a query was not passed in
    whereClause = ""
Else
    whereClause = BuildSQLWhere(xmlWhere) 'convert the query to valid sql
End If

'initialize the sql statement that will query for the titles
strSQL = "select title_id,title,type,price,ytd_sales,notes,pubdate from titles " & whereClause

Call NewRecordSet 'create a record set
'Set the cursor location
m_adoRs.CursorLocation = adUseClient
'open the recordset
m_adoRs.Open strSQL, m_dbConnection, adOpenForwardOnly, adLockReadOnly, adCmdText

'disconnect the recordset
Set m_adoRs.ActiveConnection = Nothing

On Error GoTo 0 'turn off error handler

'Close the database to free the connection
Call CloseDatabase

If m_adoRs.EOF Then
    GetTitlesXML = "" 'query did not return any titles
Else
    If lMSADO Then
        GetTitlesXML = msado(m_adoRs) 'convert the recordset to Microsoftado-->xml
    Else
        GetTitlesXML = ADOToXML(m_adoRs, True) 'convert the ado recordset to custom xml
    End If
End If

```

```

'Close the recordset
Call CloseRecordset

Exit Function

SQLErr:
    Call logerror(strSQL)

End Function

Private Function ADOTOXML(tmprs As ADODB.Recordset, tmpMP As Boolean) As String

Dim adoFields As ADODB.Fields 'set up a collectio to hold the fields
Dim adoField As ADODB.Field 'used to retrieve each field from the collection
Dim xmlDoc As msxml2.DOMDocument30
Dim tmpLine As String 'holds the xml representation of each book
Dim tmpXML As String 'used to concatenate each line of xml
Dim i As Integer

If tmprs.EOF Then 'no titles returned by query
    ADOTOXML = ""
    Exit Function
Else
    Set adoFields = tmprs.Fields 'create the collection of fields
End If

tmpXML = "<TITLES>" 'all books will be wrapped in a <TITLES> tag

Do Until tmprs.EOF 'loop through each title in the recordset
    i = 0 ' i is an index to the ado field its initialized to 0 so that first field will be field(0)
    tmpLine = "<BOOK>" & tmprs("title") & vbCrLf
    For Each adoField In adoFields 'loop through all the fields
        'build the xml <FIELD> tag and its attributes for the current field
        tmpLine = tmpLine & "<FIELD "
        tmpLine = tmpLine & "prettyname=""" & gprettyName(i) & """" "
        tmpLine = tmpLine & "tablename=""" & gtableName(i) & """" gcolumnName=""" & adoField.Name & """" "
        tmpLine = tmpLine & "datatype=""" & gdatatype(i) & """" gfilter="""""
        tmpLine = tmpLine & ">" & adoField.Value
        tmpLine = tmpLine & "</FIELD>" & vbCrLf
        i = i + 1 'point index to next field
    Next
    tmpXML = tmpXML & tmpLine & "</BOOK>" & vbCrLf 'end the book tag after last field
    tmprs.MoveNext 'next title
Loop
Set adoField = Nothing 'destroy the field object
Set adoFields = Nothing 'destroy the fields collection

tmpXML= tmpXML & "<?xml version="1.0"?></TITLES>" & vbCrLf 'end the string with the ending </TITLES> tag
'at this point I could just return this string back
'load the xml string into a DOM just to show the loadxml method
'also this will test to make sure the string is well-formed
Set xmlDoc = New msxml2.DOMDocument30 'create a xmlDOM
xmlDoc.async = False 'wait for document to load
xmlDoc.validateOnParse = False 'do not validate against a schema
xmlDoc.loadXML(tmpXML) 'load the string into the DOM
On Error Resume Next 'if the file bellow does not exist I will get an error when I try to kill it
Kill("c:\temp\custom.xml") 'erase the file if it exists
On Error GoTo 0 'set error handler to abort on error
xmlDoc.save ("c:\temp\custom.xml") 'save the xml to a file
ADOTOXML=xmlDoc.xml 'return the xml string
Set xmlDoc=Nothing 'destroy the xml DOM

End Function

Private Function msado(tmprs As ADODB.Recordset) As String

Dim xmlDoc As msxml2.DOMDocument30
On Error Resume Next 'if the file bellow does not exist I will get an error when I try to kill it
Kill ("c:\temp\msado.xml") 'erase the file if it exists
On Error GoTo 0 'set error handler to abort on error
tmprs.save "c:\temp\msado.xml", adPersistXML 'save the xml to a file
Set xmlDoc = New msxml2.DOMDocument30 'create a xml DOM
xmlDoc.async = False 'wait for document to load
xmlDoc.validateOnParse = False 'do not validate against a schema
xmlDoc.Load ("C:\temp\msado.xml") 'load the file into the DOM
msado = xmlDoc.xml 'return the xml string
Set xmlDoc = Nothing 'destroy the xml DOM

End Function

Private Sub CloseRecordset()

'Close recordset objects and dereference them m_adoRs.Close Set
m_adoRs =Nothing

End Sub

```


Private Sub NewRecordSet()

```
Set m_adoRs= Nothing 'Close recordset objects and dereference them
Set m_adoRs=New ADODB.Recordset
```

End Sub**Private Sub CloseDatabase()**

```
'Close database objects and dereference them
m_dbConnection.Close
Set m_dbConnection =Nothing
```

End Sub**Private Sub logerror(errSQL As String)**

```
Dim hFile As Integer
Dim expFile As String
```

```
On Error GoTo 0
gnErrNum = Err.Number
gstrErrDesc =Err.Description
gstrErrSrc = Err.Source Set
m_adoErrors = m_dbConnection.Errors

For Each m_adoErr In m_adoErrors
    gstrADOError = m_adoErr.Description & "," & CStr(m_adoErr.NativeError)
    _ & "," & CStr(m_adoErr.Number) & "," &
    m_adoErr.Source _ & "," & CStr(m_adoErr.SQLState)
Next
```

```
hFile =FreeFile
If Len(strLogPath) = 0 Then
    strLogPath = "C:\temp\"
End If
expFile = strLogPath & strAppName & ".err"
Open expFile For Append As #hFile
```

```
Print #hFile,"*****"
Print #hFile, Now()
Print#hFile, "*****"
Print #hFile,"Subroutine: " & tmpPro
Print #hFile, "Error Number:" & gnErrNum
Print#hFile, "Error Description: " & gstrErrDesc
Print #hFile, "Error Source:" & gstrErrSrc
Print #hFile, "Ado error String: " & gstrADOError
Print #hFile, "Bad SQL: " & errSQL
Close #hFile
```

End Sub**Private Sub Class_Initialize()**

```
strVersion = "xmlControl Version 1.1"
'title_id,title,type,price,ytd_sales,notes,pubdate
gtableName(0) = "titles"
gcolumnName(0) = "title_id"
gprettyName(0) = "Title Identification Number"
gdatatype(0) = "number"
gfilter(0) = ""
gtableName(1) = "titles"
gcolumnName(1) = "title"
gprettyName(1) = "Title of the Book"
gdatatype(1) = "text"
gfilter(1) = ""
gtableName(2) = "titles"
gcolumnName(2) = "type"
gprettyName(2) = "Type of Book"
gdatatype(2) = "text"
gfilter(2) = ""
gtableName(3) = "titles"
gcolumnName(3) = "price"
gprettyName(3) = "Price of the Book"
gdatatype(3) = "number"
gfilter(3) = ""
gtableName(4) = "titles"
gcolumnName(4) = "ytd_sales"
gprettyName(4) = "Year to date sales"
gdatatype(4) = "number"
gfilter(4) = ""
gtableName(5) = "titles"
gcolumnName(5) = "notes"
gprettyName(5) = "Notes about the book"
gdatatype(5) = "memo"
gfilter(5) = ""
gtableName(6) = "titles"
gcolumnName(6) = "pubdate"
```

```

gprettyName(6) = "Date Published"
gdatatype(6) = "date"
gfilter(6) = ""

```

End Sub

Listing 4 - VB application to test WebClass

```

Private Sub Command1_Click()

Dim objWC As xmlControl
Dim xml As String
Set objWC = New xmlControl
objWC.strDatabase = "pubs"
objWC.strServer = "ltweb"
objWC.strUser = "sa"
objWC.strPassword = ""
objWC.lMSADO = Option2.Value
objWC.strAppName = "Article1"
Text1.Text = objWC.getTitlesXML


```

End Sub

Listing 5 - Active server page to test WebClass

```

<%@ Language=VBScript %>
<%
    'Use the WebClass ActiveX control to return xml to the browser
    ,
    'before the page will operate the WebClass.dll must be registered on the web server
    'The microsoft xml parser version 3.0 (msxml3.dll) must be registered also
    'download msxml3.exe and install
    ,

set objWC = Server.CreateObject("WebClass.xmlControl")
objWC.strDatabase = "pubs"
objWC.strServer = "ltweb"  'replace ltweb with your sql server name
objWC.strUser ="sa"  'replace sa with your sql user name
objWC.strPassword=""  'enter your sql password here
objWC.strAppName="Article1"
objWC.lMSADO=false  'true will return microsoft ado-->xml
    'false will return custom xml
    ,

Response.ContentType="text/xml"  'tell browser to expect xml
Response.write objWC.getTitlesXML  'get the xml and display it
    ,

set objWC=nothing  'destroy the object
%>

```

[Download the source code for this article](#)

Copyright © 2002-2007 Simple Talk Publishing. All Rights Reserved. [Privacy Policy](#). [Terms of Use](#)