

WORLD programmers'

The UK's
definitive
programming
guide

This month's Programmers' World

VISUAL BASIC WORKSHOP		JAVA WORKSHOP	
Using XML	194	Java Server pages	203
DELPHI WORKSHOP		PERL	
Translating languages	196	Serving up HTML	206
C++ WORKSHOP		WILF'S WORKSHOP	
Checking your e-mail	199	Finite State Machines	208

What XML is and how you can use it



PART ONE

Need to share data between applications but getting bogged down by incompatible data formats? XML is the solution...

If your e-business isn't XMLing its Java applets, you're in the dustbin of history. But not to worry, if you've seen HTML, XML will seem familiar.

Where HTML has 'evolved', XML was designed rigorously from the start. In the process, most of the baroque additions foisted on HTML's already shaky foundations by Microsoft and Netscape have been eliminated. Best of all, you can invent your own tags – you're not limited to the rather miserable set that HTML provides.

Just the data, please

The purpose of XML is to separate out the

data from the way it is displayed. XML will tell you all about a customer's order, for example but not how to lay it out on a screen. It's this separation of form from function that gives XML its power. Different applications can read (or 'parse' in XML lingo) the data, then present the data in whatever way suits them.

XML is so flexible that you'll see examples of it used for customer orders, food recipes – even plays and books. But the rules are simple – see *XML's basic principles*. Apart from one or two finer points, those four rules are about it!

Provided you stick to these rules, you can make up any piece of XML you choose and it will be accepted by an XML parser.



SEE OUR **SUPERDISC**

PATH: \handson\files\vbwkshp

Of course, the parser doesn't know what to do with it – that must be handled somewhere else.

Putting it to work

That's XML in a nutshell. But apart from e-commerce, what can it be used for?

Any place where data needs to be formatted differently for different applications, XML helps. If your data's in XML, generating a set of HTML pages for normal browsers and another for WAB-enabled devices is easy.

But to get the hang of it, we'll build a basic XML viewer. We'll deny our nobler programming instincts and do it the easy way. Internet Explorer 5 has an XML parser included. If you haven't got IE5 and can't (or don't want to) move to IE5, we've included the XML Parser control on the *SuperDisc*, along with the rest of the project files.

We'll look at displaying the following XML fragment (in the file 'test.htm'):

```
<HTML testattrib="hello">
  <BODY>
    <DIV>Some div text</DIV>
    <H1>A heading</H1>
    <P>Some text
      <P style='margin-left:1cm'>someindented text</P>
    </P>
  </BODY>
</HTML>
```

That's HTML, you cry! But it's also correct XML, and using HTML tags makes it easier

FIRST STEPS

XML's basic principles

All XML must conform to a simple but rigorous set of rules:

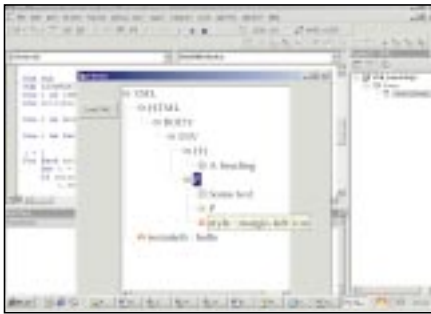
1 All XML 'tags' must be enclosed in bracket pairs and each start tag must have a matching end tag. For example, <NAME> and </NAME> are good start and end tags. If they look like HTML tags, the coincidence is entirely intentional.

2 Tag pairs must be properly nested. '<DIV><P>roses are red</P></DIV>' is great, but '<DIV><P>violets are blue</DIV></P>' isn't. Both will (probably) display ok with a browser, but an XML parser will reject the second one. You can get away with sloppy coding in HTML, with XML you can't.

3 Empty tag pairs can use an abbreviated form. The pair <P></P> can be expressed as <P/>. This turns out to have a handy side effect. Some

HTML tags don't come in pairs,
 for example. If you want to embed HTML text in XML – a useful thing to do – you can say
 to generate a line break.

4 A tag can have a set of attributes. In HTML, you can have something like '<TABLE border=1>'. The 'border' attribute modifies the way the browser displays the table. Similarly in XML, you can also have attributes: <PERSON name="fred">blah</PERSON>. Here 'name' is an attribute. Unlike HTML, all attributes, even numbers, must be quoted. Also, while XML is case sensitive-<Name> and <NAME> aren't the same – an attribute's case doesn't matter. We tend to put tags in upper case and attributes in lower case, but that's just habit.



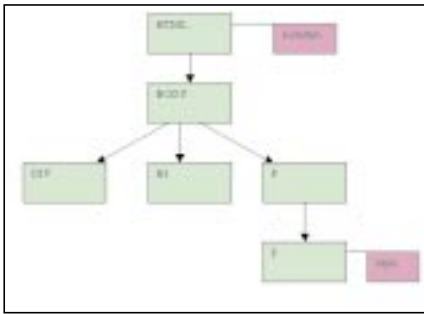
Here's the result of displaying a HTML tree. The XML nodes are represented by a folder, the terminal text by a simple file icon and attributes by an 'A'.

to see the effect. The first thing to note is that XML, like HTML, is a tree based structure. We've indented the example in order to bring this out.

Open the project XML.VBP and register the XML COM object, 'Microsoft XML, version 2' on your machine. You can then load the XML file like this:

```
Dim xmldoc As New DOMDocument, pe
As IXMLDOMParseError
If xmldoc.Load(App.Path &
"\test.htm") = False Then
    Debug.Print "can't open test
file"
Set pe = xmldoc.parseError
End If
```

This does all the parsing and error checking for you. If there's a problem, you can use the IXMLDOMParseError object to find out what went wrong. Even better, you can use Visual Basic's Watch window to poke around inside the XML document object, DOMDocument. Among a load of things that don't make very much sense (XML documents are internally quite complicated) there's a set of 'childNodes'. These hold XML nodes – IXMLDOMNodes – which correspond to the nodes of the XML tree. Each node corresponds to an XML tag, so the <BODY> tag above will have a corresponding IXMLDOMNode in the DOMDocument object.



Like HTML, XML is tree structured. Unlike HTML, XML tags must be properly nested. So while all XML documents can be considered (in principal at any rate) correct HTML documents, the converse doesn't apply.

Walking the tree

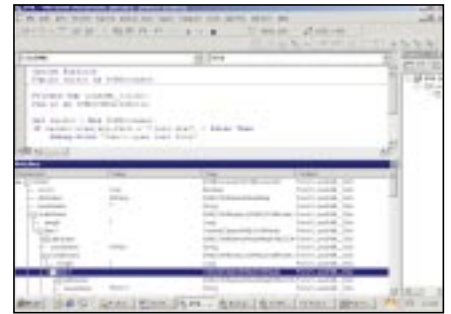
We want to produce a simple visual representation of the XML tree. The best way to this is to use a TreeView control (introduced in *Visual Basic Workshop* 163) and to fill the control's tree with information from the XML document's IXMLDOMNodes. We'll scan down the XML tree (recursively, naturally) populating the control as we go.

If an IXMLDOMNode just contained a simple piece of text, all would be very straightforward. But an IXMLDOMNode contains a whole load of other things as well: node name, XML text, plain text and the attributes to name just a few.

Some items are useful for doing some jobs, others containing more or less the same information are better suited for doing other things. We'll use the 'nodeName' to represent the name of the node as displayed in the TreeView control, and the 'Text' field to display a terminal or end node. There are several others you can use as well 'nodeTypedValue' and 'xml', for example.

Take a look at the code that loads the TreeView control, AddXMLNodes. It scans the a list of XML nodes passed to it as the 'childNodes':

```
For Each xmlnode In xmlodelist
    If xmlnode.childNodes.Length
    <> 0 Then
        ' it's a real XML node
        t.Text = xmlnode.nodeName
```



One of the nice things about Visual Basic is the Watch window. With it you can inspect an XML document, moving down the XML tree and examining every node.

```
AddXMLNodes xmlnode.
childNodes, level & ". " & i
Else
    ' it's a terminal node
    t.Text = xmlnode.Text
End If
Next
```

There are essentially two types of node – a 'real' XML node, which is a mini XML tree in its own right and a 'terminal' node, which doesn't contain any more XML tags. If a real XML node is detected, the TreeView control gets a folder icon which can be opened up to reveal the lower XML nodes. If, on the other hand, it's a terminal node, then the TreeView control gets a text file icon and you won't be able to open it up further.

Attributes are a little peculiar. An XML attribute is intended to be simple text, qualifying the XML tag in some way. You can't embed tags in attribute text, so the following is illegal:

```
<TAG myattrib="<P>hello,
world</P>">
```

Attributes are handled slightly differently than standard XML nodes. You have to test whether the attribute exists at all and if there are any attributes there. The attribute list is scanned in just the same way as the childNode list, but with the exception that we don't have to bother about recursion. Since attributes can only contain plain text, the node is 'terminal' in the above sense.

DTD IN ACTION

Structure of an XML file

Every language has its own syntax. XML's is set down in a Data Type Dictionary, or DTD

The meaning of a piece of XML is determined first by the 'grammar' and second by the content. As in politician-speak, you can have a perfectly grammatical sentence which is meaningless.

XML grammar laid out in a separate file called a DTD. This gives the rules for the XML tags. A DTD could specify that:

```
<HTML><BODY></BODY></HTML>
```

is correct, but that the following isn't:

```
<BODY><HTML></HTML></BODY>
```

Both conform to our four rules, and so are perfectly reasonable pieces of XML. It's only the DTD that says the second is wrong. So why use a DTD? In XML you can make up whatever tag you want. They might make sense you to, but will probably be meaningless to someone else. The DTD puts some order back into XML, helping others use your data by letting them know your vocabulary and syntax. Some 'validating' parsers complain if the XML they're reading isn't linked to a DTD. Luckily for us, IE5's doesn't.

But when you're experimenting with XML they're more of a hindrance than a help. We won't need to use them initially.

A more stylish walk

Feeling lost in a maze of twisty IXMDOMNodes? There's a more powerful way of extracting information from an XML document tree. It involves yet another emerging technology, XSL.

XSL (eXtensible Stylesheet Language) is, at its simplest, a technique for transforming one XML tree into another. But beware – XSL has nothing whatsoever to do with HTML's CSS or cascading style sheets. They're entirely different beasts: CSS stylesheets let you alter the formatting and layout of HTML pages; XSL style sheets are far more powerful, letting you transform and generate entirely new chunks of HTML from an XML core. **PCP**

NEXT MONTH

Turning your XML data on its head using XSL.