# Using XML And XSLT In Delphi

*by Craig Murphy*

As most website builders will know, creating a website that looks good in the plethora of browsers now available is not an easy task. Frequently we find ourselves maintaining two (or more) versions of our sites: one for Internet Explorer, the other for Netscape. With the advent of micro-browsers in PDAs and the sheer profusion of mobile telephones, it seems that managing a website is doomed to become even more complex. Thankfully, combining XML and XSLT offers light at the end of the tunnel. XML allows us to streamline our website content, while XSLT makes the presentation of the content more manageable.

Equally, we're all used to deploying our traditional applications with a collection of reports. Some of these reports will be standard, some defined by the customer, but they all require some sort of reporting engine. When changes to those reports are required, often a new executable is built and deployed. XML and XSLT can help us alleviate this requirement. After all it is much easier to deploy XSLT files: there's no recompile required and they are nothing more than plain text files.

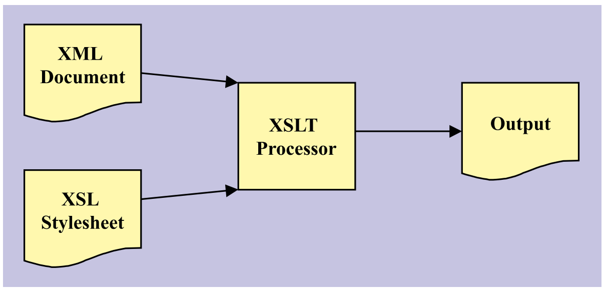
The idea of creating HTML-based reports is not new; however, there are still change and configuration management issues associated with HTML-based reports. Often the HTML tags are embedded in an executable somewhere, and there is some presentation logic embedded in the code. XML and XSLT allow us to take HTML reporting a step further: true separation of data (content) and presentation (style).

So, if you thought that XML and XSLT was purely an internet thing, read on: I will demonstrate how we can put these languages to good use in our Delphi applications. In this article I will explain how we can use XML and XSLT as a means of producing traditional reports and how they can simplify website creation.

## Introduction

The e**X**tensible **M**arkup **L**anguage (XML) has taken the world by storm: it is becoming prevalent in most aspects of computing today. The e**X**tensible **S**tylesheet **L**anguage for **T**ransformation (XSLT) provides us with a means of presenting an XML document to the user. XSLT is more than a presentation engine: it allows us to sort and filter XML too.

However, XML is about how data is represented: there is no mention of how it should be presented or displayed to the user. Whilst it is possible to programmatically work with XML and then present it to the user, this is an approach that is not easily extensible and can often be platform specific. XSLT addresses this problem by keeping the presentation (the styling) separate from the XML (the data).



➤ *Figure 1: The XSLT Processing Model.*

XML and XSLT are W3C (World Wide Web Consortium) recommendations. The full W3C recommendations for XML and XSLT are freely available: there are URLs in the *Resources* section at the end of this article. The W3C is an organisation responsible for orchestrating internet standards. Don't let the 'internet' bit put you off!

## What Is XML?

I am sure we are all familiar with some XML syntax; however, for the sake of completeness, I'll cover the basics. Listing 1 presents a simple XML document that provides enough XML syntax to get us started.

The first line of an XML document is a processing instruction, identified by the `<?` and `?>`. All XML documents have a root node, or root element, often referred to as the *document element*. In this example it's `<employees>`. Inside the root node can be any number of *child* nodes or elements. In this case, there's just one: `<employee>`. The `<employee>` element (or node) has an attribute `no`, and some child nodes: `<name>`, `<office>` and `<car>`. The element `<car>` has two attributes: `<reg>` and `<model>`. All elements have a start tag (`<employee>`) and an end tag (`</employee>`). However, some elements, like `<car>`, can be considered empty elements, hence the closing `/` after the last attribute.

The application/object that 'loads' or manages an XML

```
<?xml version="1.0"?>
<!-- this is a comment -->
<employees>
  <employee no="1">
    <name>Frank Butcher</name>
    <office>Walford</office>
    <car reg="R872 BFS" model="Vectra" />
  </employee>
</employees>
```

➤ *Above: Listing 1*

➤ *Below: Listing 2*

```
<?xml version="1.0"?>
<message>The Delphi Magazine</message>
```

```
1: <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.0">
2:
3: <xsl:template match="/">
4: <xsl:value-of select ="." />
5: </xsl:template>
6:
7: </xsl:stylesheet>
```

➤ *Above: Listing 3*

➤ *Below: Listing 4*

```
<itec:invoice
    xmlns:invoice="http://itec.co.uk/invoice">
  <amount>100</amount>
</itec:invoice>
<tdmweb:invoice
    xmlns:invoice="http://tdmweb.co.uk/invoice">
  <amount>100</amount>
</tdmweb:invoice>
```

document is known as an XML Parser.

## What Is XSLT?

XSLT is a language that allows us to alter (or transform) an existing source XML document into another destination document. The destination document need not be XML; it is perfectly possible to transform an XML document into an HTML document that can be rendered by a web browser. The application/object that performs the XSLT transformation is known as an XSLT Processor.

If this process is to work, we need to provide an XSLT Processor with a source XML document and an XSL stylesheet. The XSLT Processor then analyses the XML document using the 'rules' specified by the XSL stylesheet. These rules typically contain instructions that describe which parts of the XML document are copied to the output document. The XSLT Processor is capable of a lot more than just copying XML from one document to another, as we'll see later in this article. Figure 1 depicts this process in simplified form.

## XSLT Processors

There are a number of good XSLT Processors available and I will demonstrate two of them. The first processor is called XT: it can be downloaded from the URL specified in the *Resources* section. XT is a command-line parser. James Clark, who happens to be the editor for the XSLT 1.0 W3C recommendation, wrote XT. Another good command-line parser is Saxon: again it is a free processor. Michael Kay, whose book about XSLT is well worth a read, wrote Saxon. Internet Explorer 5 has both an XML Parser and XSLT Processor available; however, the XSLT Processor does not follow the XSLT recommendation (there is more about using IE5 later in this article).

## XSLT: A Simple Example

XSLT is itself XML and as such can be loaded by an XML parser. Thus, you may find that some XSLT Processors require the string `<?xml version="1.0"?>` as their first line. The processors I have used here do not enforce this, so I have chosen not to include it. Listing 2 presents a simple XML document. Listing 3 is our first look at an XSL stylesheet. XSLT is a declarative, rules-based programming language in which the order of the rules or the 'code' does not actually matter.

Analysing Listing 3 line-by-line, Line 1 identifies that the document is a stylesheet: `xmlns:xsl` is an attribute whose value points to a URL where there is information defining XML elements that start with `xsl:`. In this case the namespace URL is resolvable, but you do not need an internet connection: the XSLT Processor will not try and connect to the URL! There's more about namespaces in the next section. Line 3 specifies a 'rule': `<xsl:template>`. The attribute match indicates that the rule is to match the 'root node' of the source XML document, in this case it will match `<message>`. Line 4 is known as an XSL instruction element, this instructs the XSLT processor to treat the matched node (known as the current node) as text, and to copy the node into the output document.

If Listing 2, an XML document, was transformed using Listing 3, an XSL stylesheet, the resulting output document would contain the string `The Delphi Magazine`.

## Namespaces Explained

We've seen that each XSLT element requires a namespace qualifier: `xsl`. The namespace `xsl` gives the XSLT processor the ability to

```
<?xml version="1.0"?>
<employees>
<employee emp_no="2">
<emp_lastname>Nelson</emp_lastname><emp_firstname>Roberto</emp_firstname>
  <emp_phoneext>250</emp_phoneext><emp_salary currency="UKP">40000</emp_salary>
</employee>
<employee emp_no="4">
<emp_lastname>Young</emp_lastname><emp_firstname>Bruce</emp_firstname>
  <emp_phoneext>233</emp_phoneext><emp_salary currency="USD">55500</emp_salary>
</employee>
</employees>
```

➤ *Above: Listing 5*

➤ *Below: Listing 6*

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:output omit-xml-declaration="yes" />
  <xsl:template match="employees">
  <HTML>
  <BODY>
  <H1>EMPLOYEE LISTING</H1><BR />
  <TABLE>
    <TR><TD>Employee number</TD><TD>Last Name</TD><TD>First Name</TD></TR>
    <xsl:for-each select="employee">
    <TR><TD><xsl:value-of select="@emp_no" /></TD>
    <TD><xsl:value-of select="emp_lastname" /></TD>
    <TD><xsl:value-of select="emp_firstname" /></TD>
    </TR>
    </xsl:for-each>
    </TABLE>
  </BODY>
  </HTML>
  </xsl:template>
</xsl:stylesheet>
```

```
procedure TForm1.btnCreateXMLClick(Sender: TObject);
var
  f : TextFile;
  i : Integer;
begin
  i := 0;
  AssignFile(f, g_AppPath + '\XT\employees.xml');
  Rewrite(f);
  writeln(f, '<?xml version="1.0"?>');
  writeln(f, '<employees>');
  writeln(f, '');
  with tblEmployee do begin
    Open;
    First;
    DisableControls;
    while not eof do begin
      write(f, '<employee emp_no="');
      write(f, FieldByName('EMPNO').AsString);
      writeln(f, '">');
      write(f, '<emp_lastname>');
      write(f, FieldByName('LASTNAME').AsString);
      write(f, '</emp_lastname>');
      write(f, '<emp_firstname>');
      write(f, FieldByName('FIRSTNAME').AsString);
      write(f, '</emp_firstname>');
      write(f, '<emp_phoneext>');
      write(f, FieldByName('PHONEEXT').AsString);
      write(f, '</emp_phoneext>');
      // For every other employee, alternate the currency...
      // ..for the sake of demonstration...
      if i mod 2 = 0 then
        write(f, '<emp_salary currency="UKP">')
      else
        write(f, '<emp_salary currency="USD">');
      write(f, FieldByName('Salary').AsString);
      writeln(f, '</emp_salary>');
      writeln(f, '</employee>');
      writeln(f, '');
      next;
      i:=i+1;
    end;
    EnableControls;
    Close;
  end;
  writeln(f, '</employees>');
  CloseFile(f);
  mXML.Lines.LoadFromFile(g_AppPath + 'XT\employees.xml');
  wbXML.Navigate(g_AppPath + 'XT\employees.xml');
end;
```

➤ *Listing 7*

recognise which elements provide XSL instructions, such as `<xsl:value-of ...>` Listing 4 provides an example XML document that uses two namespaces.

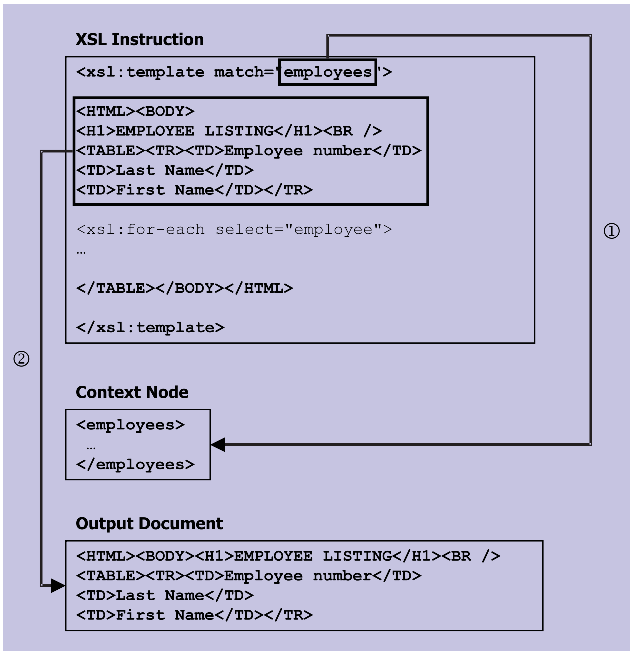
By prefixing elements with a namespace qualifier, for example `itec` or `tdmweb`, we can prevent element name conflicts. They also allow us to assume some semantic meaning, for example does the `tdmweb` amount include VAT or not? On the surface, we don't know and we cannot tell. However, because we have a namespace, we can 'work within the notional context of http://tdmweb.co.uk/invoice', and can thereby adhere to the rules for a tdmweb invoice. The same can be said for the `itec` namespace.

Namespaces use URLs because we can guarantee that they are unique. The URLs used don't have to be resolvable either; that is, there can be nothing at the other end. It is only 'meaning' that is derived from the namespace URL. However, in the future, we might see something 'more solid' appear at the end of namespace URLs.

So, in a nutshell, namespaces allow us to process two invoices and two amounts, each interpreted slightly differently.

## XSLT: A Worked Example

Listing 5 presents some of the `Employee` table from `DBDEMOS` as an XML document. For the purposes of this article, I have added an extra field `currency` to each employee record. I will be using the `currency` field to demonstrate the filtering and selection capabilities of XSLT. The Delphi code to create the XML document is part of the example that is included on the companion disk and is shown in Listing 7.


XSL Instruction
<xsl:template match="employees">
<HTML><BODY>
<H1>EMPLOYEE LISTING</H1><BR />
<TABLE><TR><TD>Employee number</TD>
<TD>Last Name</TD>
<TD>First Name</TD></TR>
<xsl:for-each select="employee">
…
</TABLE></BODY></HTML>
</xsl:template>
①
②
Context Node
<employees>
…
</employees>
Output Document
<HTML><BODY><H1>EMPLOYEE LISTING</H1><BR />
<TABLE><TR><TD>Employee number</TD>
<TD>Last Name</TD>
<TD>First Name</TD></TR>

➤ *Figure 2: Processing an XSL instruction: stage 1, identify the context node; stage 2, copy non-xsl elements to the output stream.*

```
<HTML>
<BODY>
<H1>EMPLOYEE LISTING</H1><BR>
<TABLE>
<TR>
  <TD>Employee number</TD>
  <TD>Last Name</TD>
  <TD>First Name</TD>
</TR>
<TR><TD>2</TD>
<TD>Nelson</TD>
<TD>Roberto</TD>
</TR>
<TR><TD>4</TD>
<TD>Young</TD>
<TD>Bruce</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

➤ *Listing 8*

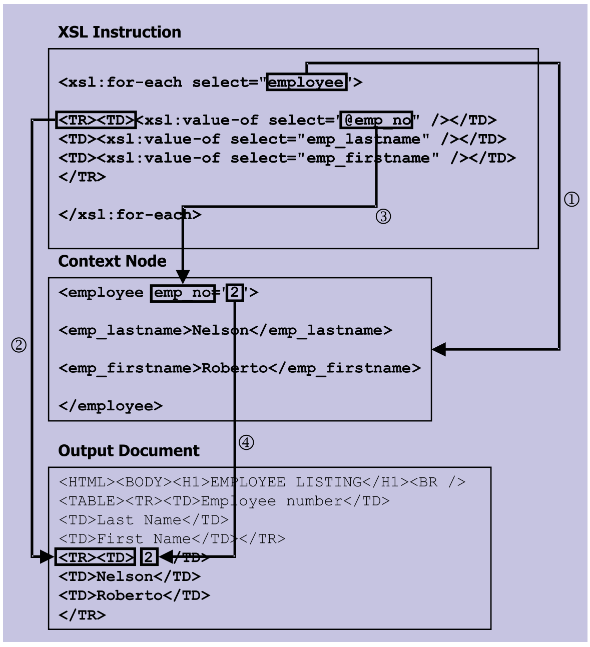
Listing 6 presents a stylesheet that is capable of transforming Listing 5 into HTML for display. Notice how the standard HTML elements, `<HTML>`, `<BODY>`, etc. are not prefixed with `xsl:`. The XSLT processor will simply copy these elements into the output document. Note, however, that empty HTML elements, like `<HR>` and `<BR>`, need to be re-written as `<HR />` and `<BR />`, ie as XML empty elements.

We have already seen how `<xsl:template>` works. In Listing 6 it selects the root node `<employees>`. With `<employees>` as the 'context' or current node, the XSLT Processor processes everything inside `<xsl:template>` until it meets a matching end element, `</xsl:template>`. Figure 2 represents this match.

The XSLT Processor then copies all elements without the `xsl:` prefix over to the output document. The first xsl instruction that the processor has to deal with is: `<xsl:for-each select="employee">`.

We know that the `<employees>` element contains a number of `<employee>` elements, therefore this xsl instruction can be read as 'for each employee do...'. Inside the `<xsl:for-each>` element, the context node becomes the first employee element, then the second, and so on. `<xsl:value-of select="@emp_no" />` instructs the processor to copy the attribute `emp_no` from the input document into the output document. Figure 3 represents this process: remember that the XSLT Processor is processing an `<xsl:for-each>` construct, so it will iterate over all the `<employee>` elements inside `<employees>`. Figure 4 represents the output when viewed in a browser. Listing 8 represents a sample from the output document.

➤ *Figure 3: Processing an <xsl:for-each> element: stage 1, identify the context node; stage 2, copy non-xsl elements to the output stream; stage 3, identify the attribute emp_no in the context node; stage 4, copy the value of the attribute emp_no to the output stream.*



```
<xsl:for-each select="employee">

<TR><TD><xsl:value-of select="@emp_no" /></TD>
<TD><xsl:value-of select="emp_lastname" /></TD>
<TD><xsl:value-of select="emp_firstname" /></TD>
</TR>

</xsl:for-each>
```

```
<employee emp_no='2'>

<emp_lastname>Nelson</emp_lastname>

<emp_firstname>Roberto</emp_firstname>

</employee>
```

```
<HTML><BODY><H1>EMPLOYEE LISTING</H1><BR />
<TABLE><TR><TD>Employee number</TD>
<TD>Last Name</TD>
<TD>First Name</TD></TR>
<TR><TD>2</TD>
<TD>Nelson</TD>
<TD>Roberto</TD>
</TR>
```

## Filtering XML Elements Using XSLT

Listing 9 demonstrates two ways in which XSLT allows us to filter XML elements. Effectively, Listing 9 is performing a query that reads: 'List all employees whose salary is greater than 50000 and whose currency is USD'.

The `select` attribute in an `<xsl:for-each>` element takes an 'expression'. At this point the XSLT recommendation draws on another recommendation: XPath. XPath provides us with a means of 'addressing the elements with an XML document'. For example, the following expression would select all the employee elements where the employee `emp_salary` element has a value greater than 50000: employee[emp_salary > 50000].

If we combine this expression in an `<xsl:for-each>` element, we have something that is fairly useful: `<xsl:for-each select="employee[emp_salary > 50000]">`

A second method of filtering out unwanted XML elements is the use of an `<xsl:if>` element. The `test` attribute in an `<xsl:if>` element takes a Boolean XPath expression. For example, the following expression would be true if the `currency` attribute of the `emp_salary` element (from within the current

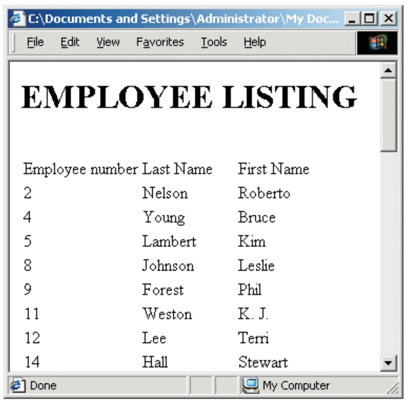
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:output omit-xml-declaration="yes" />
  <xsl:template match="employees">
  <HTML>
  <BODY>
  <H1>EMPLOYEE LISTING</H1><BR />
    <TABLE>
    <TR><TD>Employee number</TD><TD>Last Name</TD>
      <TD>First Name</TD><TD>Salary</TD></TR>
    <xsl:for-each select="employee[emp_salary > 50000]">
    <xsl:if test="emp_salary[@currency = 'USD']">
    <TR><TD><xsl:value-of select="@emp_no" /></TD>
    <TD><xsl:value-of select="emp_lastname" /></TD>
    <TD><xsl:value-of select="emp_firstname" /></TD>
    <TD><xsl:value-of select="emp_salary" /></TD>
    </TR>
    </xsl:if>
    </xsl:for-each>
    </TABLE>
  </BODY>
  </HTML>
  </xsl:template>
</xsl:stylesheet>
```

➤ *Above : Listing 9*

➤ *Below: Listing 10*

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:output omit-xml-declaration="yes" />
  <xsl:template match="employees">
  <HTML>
  <BODY>
  <H1>EMPLOYEE LISTING</H1><BR />
    <TABLE>
    <TR><TD>Employee number</TD><TD>Last Name</TD>
      <TD>First Name</TD><TD>Currency</TD></TR>
    <xsl:for-each select="employee">
    <xsl:sort select="emp_lastname" order="ascending" />
    <xsl:if test="emp_salary[@currency = 'UKP']">
    <TR><TD><xsl:value-of select="@emp_no" /></TD>
    <TD><xsl:value-of select="emp_lastname" /></TD>
    <TD><xsl:value-of select="emp_firstname" /></TD>
    <TD><xsl:value-of select="emp_salary/@currency" /></TD>
    </TR>
    </xsl:if>
    </xsl:for-each>
    </TABLE>
  </BODY>
  </HTML>
  </xsl:template>
</xsl:stylesheet>
```

➤ *Figure 4: An XSLT transformation as viewed in a browser.*

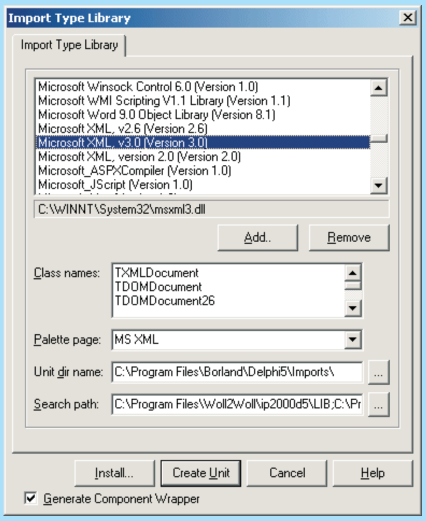


EMPLOYEE LISTING

| Employee number | Last Name | First Name |
|---|---|---|
| 2 | Nelson | Roberto |
| 4 | Young | Bruce |
| 5 | Lambert | Kim |
| 8 | Johnson | Leslie |
| 9 | Forest | Phil |
| 11 | Weston | K. J. |
| 12 | Lee | Terri |
| 14 | Hall | Stewart |

### Importing The MSXML Type Library

From the Project menu, choose Import Type Library, navigate through the list of type libraries until you find Microsoft XML, v3.0. Figure 5 should look familiar. Clicking on the `Create Unit` button will create a Delphi implementation of the type library, you should find it in your C:\Program Files\Borland\Delphi5\Imports directory. Add the newly created unit, MSXML2_TLB.pas, to your project. You are now ready to start using the Microsoft Parser.



**Import Type Library**

Import Type Library

```
Microsoft Winsock Control 6.0 (Version 1.0)
Microsoft WMI Scripting V1.1 Library (Version 1.1)
Microsoft Word 9.0 Object Library (Version 8.1)
Microsoft XML, v2.6 (Version 2.6)
Microsoft XML, v3.0 (Version 3.0)
Microsoft XML, version 2.0 (Version 2.0)
Microsoft_ASPXCompiler (Version 1.0)
Microsoft_JScript (Version 1.0)
```

C:\WINNT\System32\msxml3.dll

Class names: TXMLDocument, TDOMDocument, TDOMDocument26

Palette page: MS XML

Unit dir name: C:\Program Files\Borland\Delphi5\Imports\

Search path: C:\Program Files\Woll2Woll\ip2000d5\LIB;C:\Pr

[✓] Generate Component Wrapper

➤ Figure 5

node) held the value `USD`: `emp_salary[@currency = 'USD']`. If we combine this expression in an `<xsl:if>` element, we have another useful filtering mechanism:

```
<xsl:if test=
 "emp_salary[@currency = 'USD']">
```

## Sorting XML Using XSLT

Listing 10 demonstrates how we can use XSLT to re-order XML elements. An `<xsl:sort>` element is powerful enough to perform a sort based on a number of criteria: order (ascending or descending), case-order (upper-first or lower-first) and data-type (text, number or user-defined). For example, the following `<xsl:sort>` element sorts the returned elements using the value of the `emp_lastname` element, in ascending order:

```
<xsl:sort select="emp_lastname"
  order="ascending" />
```

## XSLT From Microsoft

Microsoft pre-empted the W3C with the release of Internet Explorer 5 during 1998. Part of the IE 5 installation, MSXML.DLL, provided programmatic access to an XML parser and an XSL processor. However, during 1999 the W3C XSLT specification changed dramatically; by 16 November 1999 the specification had become an official W3C recommendation. The differences between the W3C recommendation and the Microsoft implementation were significant, so significant that the Microsoft implementation is frequently referred to as XSL98.

However, Microsoft recognised that things had to change, so they embarked on the development of a conforming XSLT Processor, known as MSXML3. To ensure that they kept the developer community active, MSXML3 was released as a series of 'web releases', or beta versions. The beta programme culminated with the release of the final version appearing on the last day of October 2000.

When you install IE 5 (and 5.5), an earlier version of MSXML is installed. XSL98 is available, but XSLT 1.0 is not. You can download MSXML 3 and install it, but you must also have IE 5.x installed.

MSXML3 (and the original version, MSXML.DLL) offers an XML Parser and an XSLT Processor as COM components. As Delphi developers we can take advantage of these free components. The downside of using IE 5 is that it must be installed on the machine that your application is deployed on. Similarly, MSXML3.DLL must be registered too. Figure 6 depicts this situation. However, if you are able to guarantee these requirements, you may take advantage of these components in your Delphi applications.

## Using MSXML In Delphi

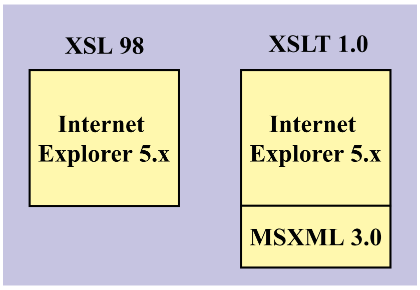
So far, all the XML/XSLT examples have been processed using the XSLT Processor XT. Whilst this is a free XSLT Processor, it does add nearly 800Kb to the size of the application being deployed. However, if you are able to guarantee that Internet Explorer 5 is installed on the machine on which you plan to deploy your application, you may use the Microsoft XML Parser and XSLT Processor.

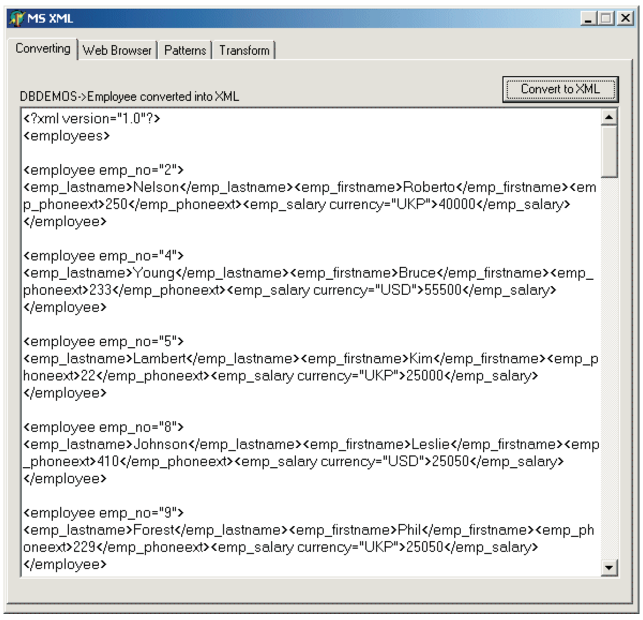
Internet Explorer makes displaying XML that has been automatically transformed that little bit easier. By adding another processing instruction to the XML file, we can instruct Internet Explorer to display XML using a specified stylesheet. Listing 11 includes processing instructions required to transform the XML using the emp_style.xsl stylesheet.

However, it is likely that you may wish to control precisely which XSL stylesheet is used: this is where Delphi fits in. I will assume that you have installed Internet Explorer 5.x, MSXML 3.0 (available from the URL in the *Resources* section) and have imported the `MSXML` type library into your Delphi project (see the sidebar for details of how to do this). Listing 12 presents the code required to perform an XSLT transform using the Microsoft offering; Figure 8 presents the results of the transform in a `TWebBrowser` component

In true Microsoft fashion, their implementation of the XML parser and XSLT processor offers a few 'non-standard' extensions. One of those extensions is a very practical `selectNodes` method. This method takes an XPath expression and returns a list of matching nodes. Listing 13 demonstrates `selectNodes` in practice; Figure 8 presents the results of a call to `selectNodes`.

➤ *Figure 6: The Internet Explorer 5.x XML/XSLT model.*

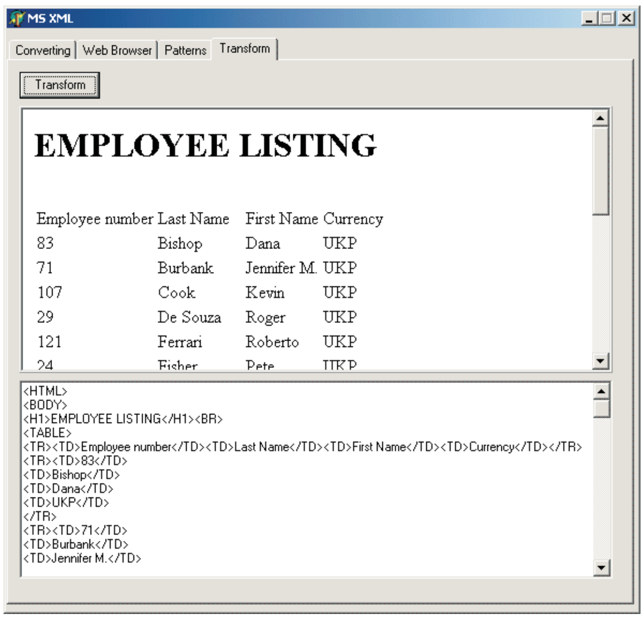




DBDEMOS->Employee converted into XML

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

<employee emp_no="2">
<emp_lastname>Nelson</emp_lastname><emp_firstname>Roberto</emp_firstname><em
p_phoneext>250</emp_phoneext><emp_salary currency="UKP">40000</emp_salary>
</employee>

<employee emp_no="4">
<emp_lastname>Young</emp_lastname><emp_firstname>Bruce</emp_firstname><emp_
phoneext>233</emp_phoneext><emp_salary currency="USD">55500</emp_salary>
</employee>

<employee emp_no="5">
<emp_lastname>Lambert</emp_lastname><emp_firstname>Kim</emp_firstname><emp_p
honeext>22</emp_phoneext><emp_salary currency="UKP">25000</emp_salary>
</employee>

<employee emp_no="8">
<emp_lastname>Johnson</emp_lastname><emp_firstname>Leslie</emp_firstname><emp
_phoneext>410</emp_phoneext><emp_salary currency="USD">25050</emp_salary>
</employee>

<employee emp_no="9">
<emp_lastname>Forest</emp_lastname><emp_firstname>Phil</emp_firstname><emp_ph
oneext>229</emp_phoneext><emp_salary currency="UKP">25050</emp_salary>
</employee>
```

➤ *Figure 7: DBDemos Employee table as XML.*



EMPLOYEE LISTING

| Employee number | Last Name | First Name | Currency |
|---|---|---|---|
| 83 | Bishop | Dana | UKP |
| 71 | Burbank | Jennifer M. | UKP |
| 107 | Cook | Kevin | UKP |
| 29 | De Souza | Roger | UKP |
| 121 | Ferrari | Roberto | UKP |
| 24 | Fisher | Pete | UKP |

```
<HTML>
<BODY>
<H1>EMPLOYEE LISTING</H1><BR>
<TABLE>
<TR><TD>Employee number</TD><TD>Last Name</TD><TD>First Name</TD><TD>Currency</TD></TR>
<TR><TD>83</TD>
<TD>Bishop</TD>
<TD>Dana</TD>
<TD>UKP</TD>
</TR>
<TR><TD>71</TD>
<TD>Burbank</TD>
<TD>Jennifer M.</TD>
```

➤ *Figure 8:  Transform with the MS XSLT Processor.*

## XSL98 And XSLT

Without MSXML3.DLL installed, IE 5 will not process a stylesheet where the root element is:

```
 <xsl:stylesheet xmlns:xsl=
   "http://www.w3.org/1999/XSL/
   Transform" version="1.0">
```

Instead, we have to specify a root element with a different xsl namespace:

```
 <xsl:stylesheet xmlns:xsl=
 "http://www.w3.org/TR/WD-xsl">
```

If XSL98 is so different from XSLT, why is it still so useful? If you are responsible for the creation of websites, then XSL98 might be of use to you.  If your web hosting company uses Windows NT or Windows 2000 with Internet Explorer 5 installed, then you can take advantage of XSL98.  The actual syntax differences between XSL98 and XSLT are minimal, so the time you invest in learning XSL98 will make the transition to XSLT that much easier.

If you are using TDMWeb to host your websites you can take advantage of XML and XSL98 for free on your sites. The VBScript file simple_transform.asp on the companion disk shows how to use the Microsoft Parser and Processor (for more on TDMWeb web hosting see www.TDMWeb.com).

If you wish to learn more about XSL98, there is a book available: *XML IE5* by Alex Homer published by Wrox Press (the ISBN is 1-861001-57-6).

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="emp_style.xsl"?>
<employees>
<employee emp_no="2">
<emp_lastname>Nelson</emp_lastname><emp_firstname>Roberto</emp_firstname>
  <emp_phoneext>250</emp_phoneext><emp_salary currency="UKP">40000</emp_salary>
</employee>
</employees>
```

➤ *Listing 11*

## Now And The Future

We often see XML mentioned in the same breath as Application-2-Application (A2A) and Business-2-Business (B2B). Apart from the obvious ease of transport that XML gives us, it's easily transformable. XSLT allows us to convert 'your invoice' into 'my invoice', and this is exactly what products like BizTalk can do for us. At the heart of BizTalk is a tool known as the BizTalk Mapper: it provides a graphical user interface that allows the creation of XSLT that transforms your invoice into my invoice. During the transformation, scripts can access databases to provide for a more accurate transformation.

Underlying BizTalk is SOAP (Simple Object Access Protocol), which provides us with a protocol for the structured exchange of data, using XML as the data representation.

## Summary

Over the course of this rather busy article, we have seen how it is possible to create HTML reports using Delphi, XML and XSLT. Hopefully the small amount of XSLT that I have shown here has whetted your appetite for more: it is a phenomenally powerful language that not only transforms the structure of XML documents, but allows a real degree of control with the provision of filtering, sorting, and computational elements. If this brief look at XSLT makes you want to dip into your wallet, I can strongly recommend the *XSLT Programmer's Reference* by Michael Kay, published by Wrox Press (ISBN 1-861003-12-9).

Craig Murphy works as an Enterprise Developer for Currie & Brown (www.currieb.com) whose primary business is quantity surveying, cost management and project management. He can be contacted via email at Craig@isleofjura.demon.co.uk

```
procedure TForm1.TransformClick(Sender: TObject);
var
  xmlEmployees, xslStyle : IXMLDOMDocument;
begin
  xmlEmployees := CoDOMDocument.Create;
  xslStyle     := CoDOMDocument.Create;
  xmlEmployees.load('employees.xml');
  xslStyle.load('empuksna.xsl');
  memoTransform.Text := xmlEmployees.transformNode(xslStyle);
  memoTransform.Lines.SaveToFile('msxml.htm');
  wbMSXML.Navigate(g_AppPath+'msxml.htm');
end;
```

➤ *Above: Listing 12*

➤ *Below: Listing 13*

```
procedure TForm1.btnApplyClick(Sender: TObject);
var
  xmlResult : IXMLDOMNodeList;
  i : Integer;
begin
  XMLResult:=g_xmlDoc.selectNodes (edtPattern.Text);
  mResults.Text:='';
  lblCount.Caption:= IntToStr(XMLResult.Length) + ' item(s) returned';
  for i:=0 to XMLResult.Length-1 do
  begin
    mResults.Lines.Add(XMLResult.Item[i].XML);
  end;
end;
```

## Resources

Two command-line XSLT processors that are discussed in this article are available. For XT see www.jclark.com/xml/xt.html, for Instant Saxon see http://users.iclway.co.uk/mhkay/saxon/instant.html. Both need the Microsoft Java VM (which is installed with Internet Explorer 4.01+).

The Microsoft XML Parser (and XSLT Processor) version 3.0 is available at http://msdn.microsoft.com/xml/general/xmlparser.asp. The Microsoft offering comes with a comprehensive HTML Help file that provides good coverage of XML and is a good XSLT reference too.

The complete W3C XML 1.0 recommendation is available at www.w3.org/TR/1998/REC-xml-19980210.

The complete W3C XSLT 1.0 recommendation is available at www.w3.org/TR/1999/REC-xslt-19991116.

If you need a native Delphi XML parser, OpenXML is a good starting point: www.philo.de/xml.

On the companion disk there is a Delphi example that demonstrates XSLT for presentation of XML, the example includes filtering and sorting examples. There is a separate example that demonstrates the use of the Microsoft XML Parser and XSLT Processor.