Using XML and XSLT with Delphi 5 and WebBroker

By: Vincent Parrett

Abstract:
This article is aimed at Delphi developers who wish to make use of XML/XSLT in their WebBroker applications. I'll present some WebBroker extensions/components which make it easier to create XSLT based WebBroker applications.

By Vincent Parrett

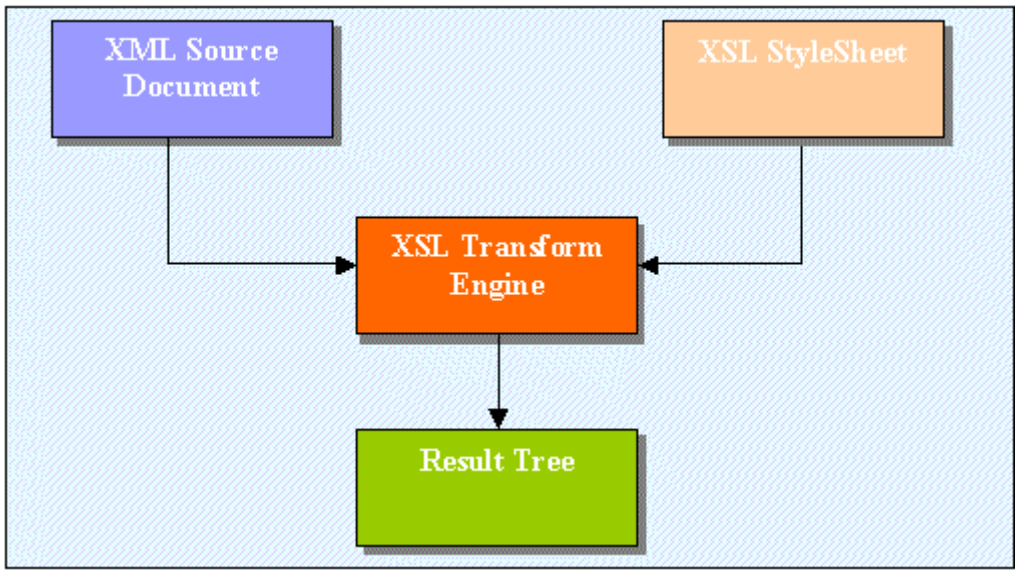
Getting Started

I'm not going to cover much of the XSLT syntax in this article , I recommend that you spend some time reading up on XSL and XSLT. I'll assume that you are at least familiar with XML/DOM, and will have at least a basic knowledge of XSLT. I have listed some reference materials at the end of this article to help you get up to speed if needed. I will also assume that you are familiar with Delphi 5.0 WebBroker applications, the example application will be ISAPI based, purely because that is my preferred WebServer application type.

What is XSL/XSLT

XSL (Extensible Stylesheet Language) is a specification from the W3C (https://www.w3.org). There are two parts to the specification, XSL Formatting and XSL Transformation (hence XSLT). We will be using the XSLT part of the specification here, the formatting part of the specification is not ratified and is not supported by any of the current crop of web browsers. Most references to XSL are generally referring to XSLT.
So what do we need to get started :

- XML Source document - a well formed xml document
- XSL Stylesheet - should conform to the XSL 1.0 spec
- XSLT Transform Engine/Processor - we will use the Microsoft XML parser/processor



The XML Source document

The xml source is a well formed xml document. In addition, it includes a stylesheet processing instruction :

```
<?xml-stylesheet type="text/xsl" href="D:\TEMP\vehicles.xsl"?>
```

This processing instruction tells XSLT Transform Processor which XSL Stylesheet to apply to the source document.

Example XML Source Document: vehicles.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="D:\TEMP\vehicles.xslt"?>
<vehicles>
  <type>Cars</type>
  <car>
    <name>Commodore</name>
    <manufacturer>Holden</manufacturer>
    <price>38,000</price>
  </car>
  <car>
    <name>Falcon</name>
    <manufacturer>Ford</manufacturer>
    <price>36,000</price>
  </car>
  <car>
    <name>Verada</name>
    <manufacturer>Mitsubishi</manufacturer>
    <price>36,500</price>
    </car>
  <car>
    <name>Avalon</name>
    <manufacturer>Toyota</manufacturer>
    <price>41,000</price>
  </car>
</vehicles>
```

The XSL Stylesheet

An XSL StyleSheet is expressed as a well-formed XML document with xsl transform instructions. The root element is where we specify that this document is a stylesheet and the xsl namespace and version. The most common use for xslt is generating an html document and that is what we will be doing here. We tell the processor that we want to output html by adding an <xsl:output> element. Note that the stylesheet is interspersed with html. The html must be written as well formed xml, this means that all tags must have closing tags.

Example XSL StyleSheet: vehicles.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="https://www.w3.org/1999/XSL/Transform" >
  <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:value-of select="vehicles/type"/>
        </TITLE>
      </HEAD>
      <BODY>
        <TABLE border="1">
          <TR>
            <TD>Name</TD>
            <TD>Manufacturer</TD>
            <TD>Price</TD>
          </TR>
          <xsl:for-each select="vehicles/car">
          <TR>
            <TD>
              <xsl:value-of select="name"/>
            </TD>
            <TD>
              <xsl:value-of select="manufacturer"/>
            </TD>
            <TD>
              <xsl:value-of select="price"/>
```

```
              </TD>
            </TR>
            </xsl:for-each>
          </TABLE>
        </BODY>
      </HTML>
    </xsl:template>
  </xsl:stylesheet>
```

The Result Tree

The output of the XSLT Processor is referred to as the Result Tree. This is because the output is usually a well formed xml document(and xml documents being tree like in structure). The above stylesheet would render the example xml source document into an html document with a table:

Output : vehicles.html

Name	Manufacturer	Price
Commodore	Holden	38,000
Falcon	Ford	36,000
Verada	Mitsubishi	36,500
Avalon	Toyota	41,000

The key feature of XSLT is that it allows you to easily separate the data from the display format. In effect what the stylesheet and xslt transformation process does is very similar to what we currently do with WebBroker. Instead of replacing custom tags in an html template with data, we use xslt to extract data from the source xml document and place into our output document.

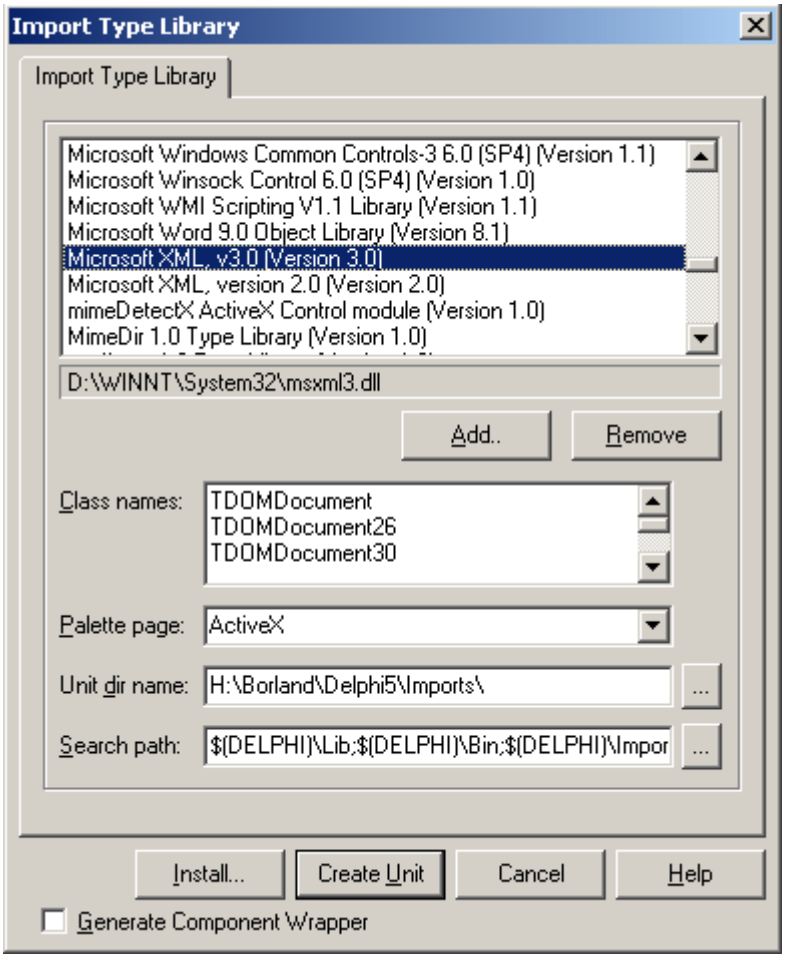
Ok, with that simple example behind us, we'll move on to using XSLT in our WebBroker applications.

XSL/XSLT Implementations

In order to use XSLT, we need a Transformation Engine/Processor. Fortunately for us, Microsoft has a pretty good one available for free. It's included with IE 5.0, or you can download it separately if you prefer not to install IE5.0. The newer MSXML3 available on the MSDN site has a more evolved implementation of XSLT that supports more of the XSLT 1.0 specification than the IE5 version, and it is that version we will be using.

Using the MSXML Parser/ Transform Engine

To use the msxml3 parser we first need to import the msxlm3 type library into Delphi. On the Project menu, select Import Type Library , and scroll down until you see the entry, Microsoft XML, v3.0 (Version 3.0). Click on the Create Unit button, making sure the 'Generate Component Wrapper' check box is unchecked (see figure 2).



This should result in a unit call MSXML2_TLB.pas in your delphi5imports directory. The Imports directory is on the Delphi 5 project path by default, if for some reason it is not then add it to your library path now.

Using the transform engine in Delphi is quite simple, below is a simple example function that makes use of the msxml transform engine to transform an xmldocument using an xsl stylesheet:

```
function DoTransform(const xml, xsl : string ): string;
var
  XMLDoc : IXMLDOMDocument;
  XSLDoc : IXMLDOMDocument;
  Template : IXSLTemplate;
  Processor : IXSLProcessor;
begin
  Result := '';
  try
    XMLDoc := CoFreeThreadedDOMDocument30.Create;
    XSLDoc := CoFreeThreadedDOMDocument30.Create;
    XMLDoc.load(xml);
    XSLDoc.load(xsl);
    Template := CoXSLTemplate30.Create;
    Template.stylesheet := XSLDoc;
    Processor := Template.createProcessor;
    Processor.input := XMLDoc;
    Processor.transform;
    result :=  Processor.output;
  finally
    XMLDoc := nil;
    XSLDoc := nil;
  end;
end;
```

This function could of course be modified to take a DOM xml document object instead of a string representation of the source xml document.

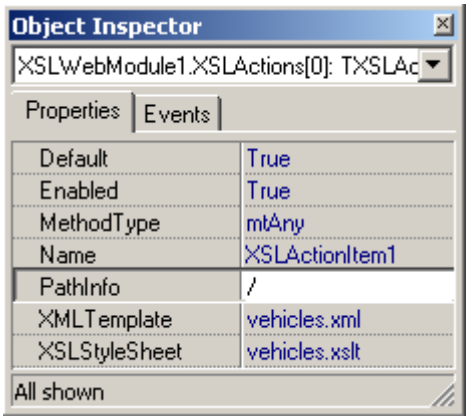
To make use of the transform code in a WebBroker application we might do something line this:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  fs : TFileStream;
  ss : TStringStream;
  xml : string;
  xsl : string;
begin

  //load our xml document from a file in this example,
  // we could of course generate the xml from other sources or build it
  // in our code

  fs := TFileStream.Create('c:\inetpub\xsldemo\test.xml',fmOpenRead     );

  // use TstringStream to get a long string version of our xml doc
  ss := TStringstream.Create('');
  ss.CopyFrom(fs,fs.Size);
  xml := ss.DataString;
  fs.Free;

  // now do the same for the xsl stylesheet
  fs := TFileStream.Create('c:\inetpub\xsldemo\test_style.xslt',fmOpenRead      );
  ss.CopyFrom(fs,fs.Size);
  xsl := ss.DataString;
  ss.Free;
  fs.Free;

  // do the transform and send the resulting html as the Response.content
  Response.Content := DoTransform(xml,xsl);
end;
```

Of course by now you might be thinking that this code looks rather ugly and long winded! This sort of code might be better off 'under the hood', so that we can concentrate on generating the xml documents with the actual data we would like to present (more about this later).

Extending WebBroker

The Delphi WebBroker architecture is really quite simple to use. We link up action items to page producers of various types, and write event handler code to replace the custom tags with data. Sometimes we might write event handler code for the ActionItem's OnAction event. Since we won't be using PageProducer components any longer, this is where we will be writing most of our code in the examples. What we need to do is extend the WebBroker architecture so that we can do the loading of xml documents and xsl stylesheets in the background, without writing any code for this in each event handler. Unfortunately the WebBroker components were not designed for inheritance, so this involves some serious copying & pasting! The result is a new TCustomWebDispatcher descendant, TXSLWebDispatcher. This dispatcher component hides the original Actions property (they are still there, just used under the hood) and exposes an XSLActions property. The XSLActions property is very similar to the original Actions property. TXSLActionItem replaces the TWebActionItem, it has some additional properties which are of interest:



Note that the producer property has gone and there are two additional properties, XMLTemplate and XSLStyleSheet.

The XMLTemplate property allows you to specify an existing xml document that will be loaded when this XSLAction is called. This property should specify just the file name of the xml file (not the full path).

The XSLStylesheet property allows you to specify an XSL stylesheet that will be used in the transform process when the action is called.

The method signature for the OnAction Event looks like this:

```
procedure TXSLWebModule1.XSLActionItem1_OnAction(
            Sender: TObject; Request: TWebRequest; Response: TWebResponse;
            Document: IXMLDOMDocument; var stylesheet: String;  var Handled: Boolean);
```

The other significant change is the method signature for the OnAction event handler. All the old parameters are there, with the addition of :

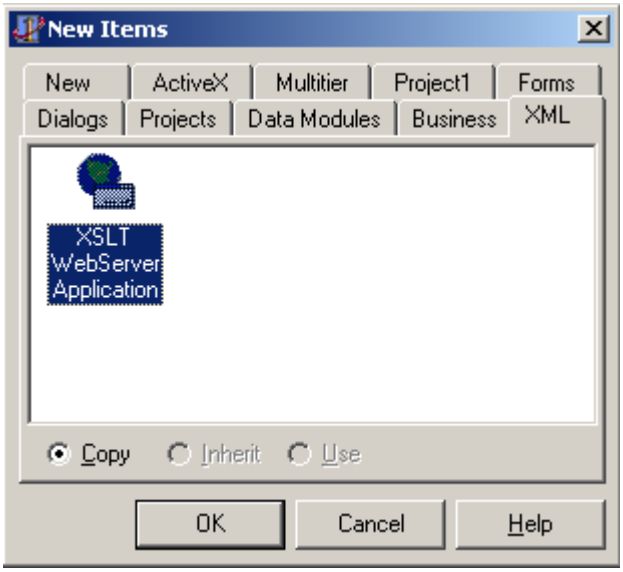
Document : IXMLDOMDocument - this DOM document object is loaded with the document specified in the XMLTemplate property of the action. If that property is blank then the Document object will contain an empty xml document.

var stylesheet : string - this parameter contains the name of the XSL stylesheet file specified in the XSLStylesheet property of the action. You can modify this if you want to process the xml with a different stylesheet. A typical example of this might be a standard error handling stylesheet.

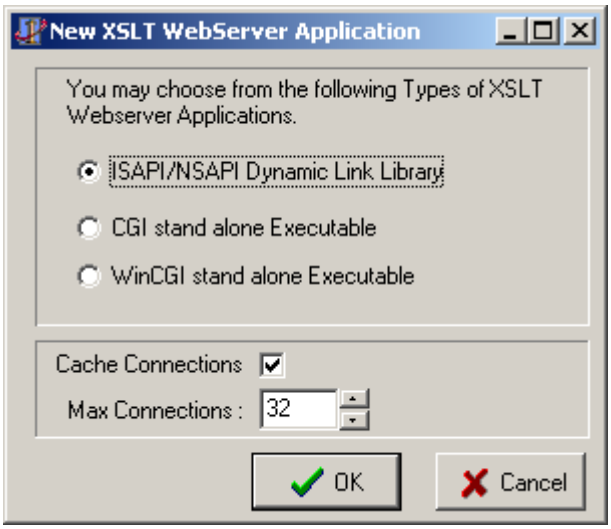
Using the WebBroker Extensions

Using the supplied installation program, the extensions should automatically be installed in Delphi 5. Make sure that Delphi is not running when you run the install program.

Once the WebBroker Extensions are installed, you should see a new tab in the New Items dialog, XML.



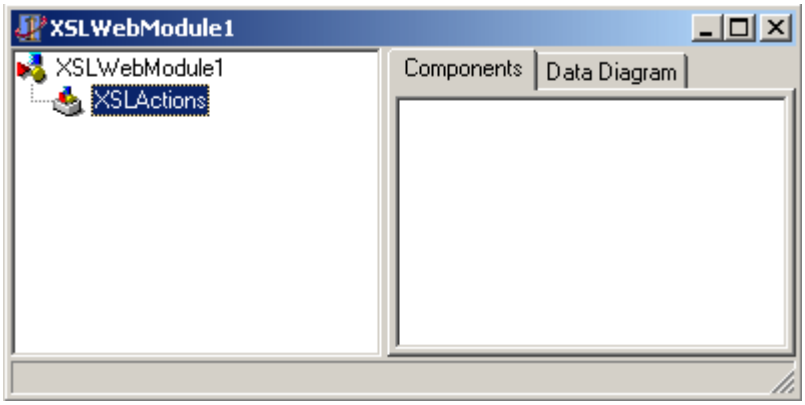
Lets get started and create an XSLT WebBroker application. Double click on the 'XSLT WebServer Application' icon, this will bring up a dialog very similar to the WebBroker application



Accepting the default options produces a project source file that looks like this:

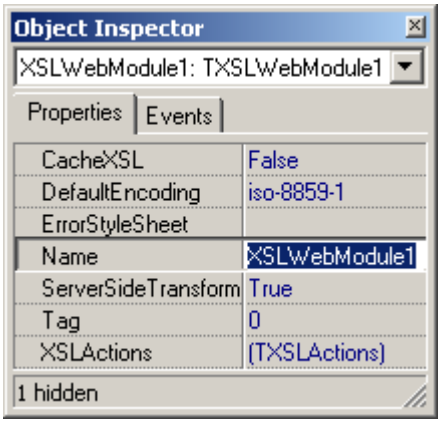
```
library Project1;

uses
  WebBroker,
  ISAPIApp,
  Unit1 in 'Unit1.pas' {XSLWebModule1: TXSLWebModule};

{$R *.RES}

exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;

begin
  Application.Initialize;
  Application.MaxConnections := 32;
  Application.CacheConnections := True;
  Application.CreateForm(TXSLWebModule1, XSLWebModule1);
  Application.Run;

end.
```

An XSLWebModule is also created by the Project Wizard :

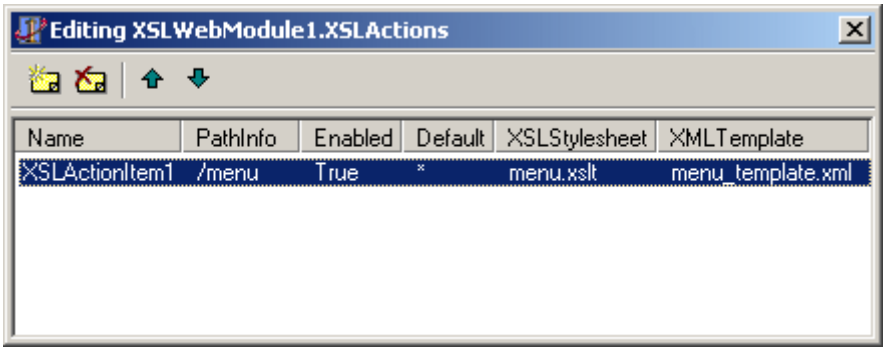


Note the WebModule does not have an Actions property, but it does have XSLActions. Assuming you have already familiar with WebBroker, then so far this should all be quite familiar too.

Now we can start developing. We'll begin by adding an action. Click on the webmodule to select it and then show the Object inspector. You should see something like this :

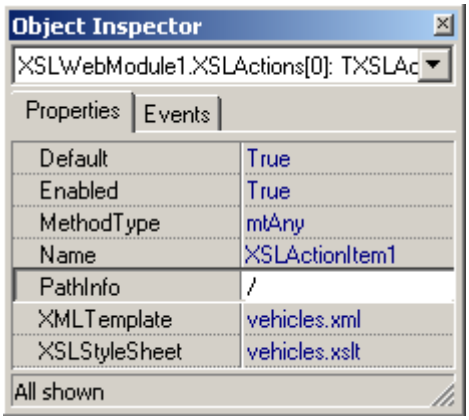


The XSLActions property is of type TXSLActions, a TCollection decendant. This means that you can use the collections property editor with it.



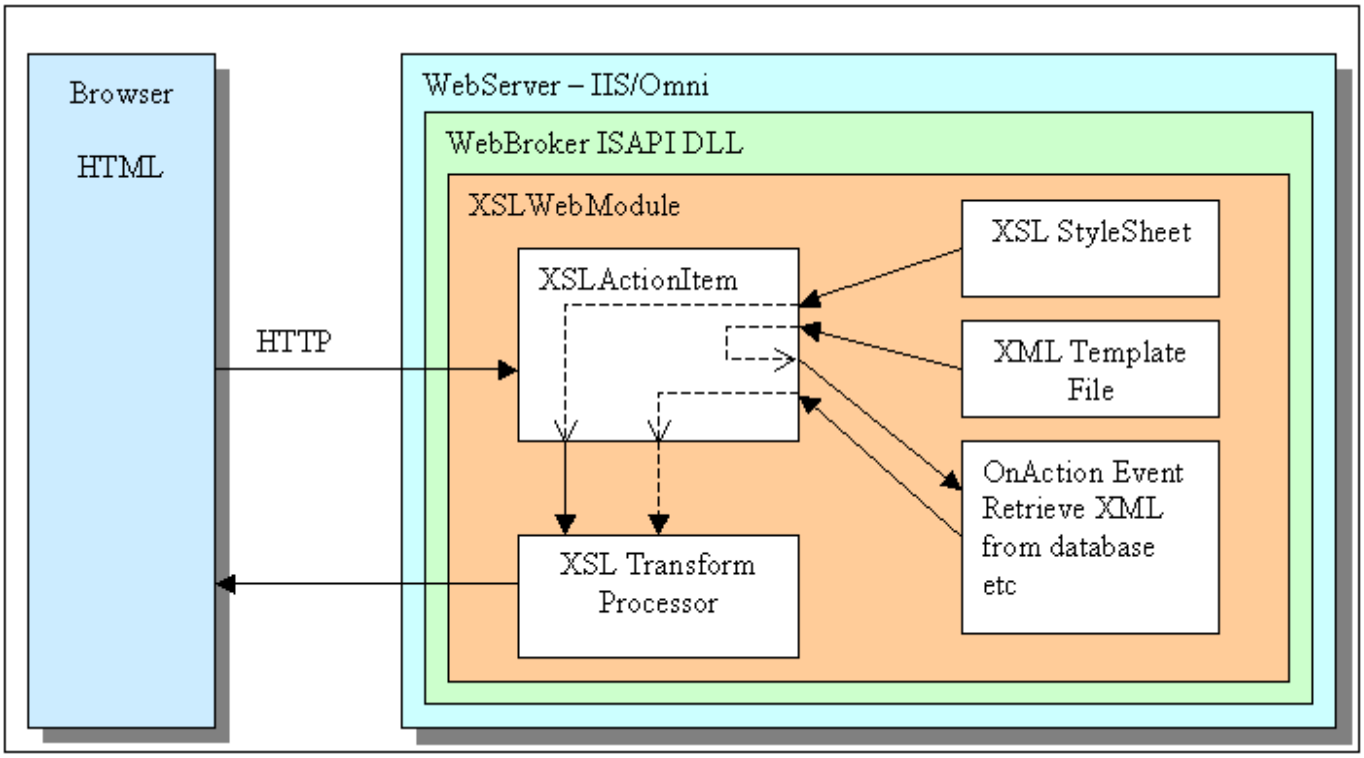
To add new XSLActions, you can either use the property editor, or right click on the XSLActions node in the tree and select Add Item

Now we can set the necessary properties on the XSLActionItem.



We'll set the pathinfo to / (the root of the application) , the XMLTemplate to vehicles.xml (the sample xml source document) and the XSLStylesheet to vehicles.xslt. When we compile the and run the dll on a webserver such as Omni (https://www.omnicron.ca) we would use a url such as : https://localhost/isapi/project1.dll/

Notice that we haven't yet written a line of code. When the application handles the request, the webmodule will load the XMLTemplate file, and then if an OnAction event handler is assigned it is called. After the event handler completes, the webmodule loads the specified xslt stylesheet for the action and passes both the xml document and the xslt stylesheet to the transform processor. The output of the processor is sent back to the client as the Response.Content.



Generating XML

So far we haven't needed to write any code, we have just transformed a static xml file (the XMLTemplate file). In real world web applications, the data is dynamic, most likely stored in a database. So how do we

generate the xml document based on our real data, stored in a database such SQLServer or Oracle, or even Microsoft Access?

The simplest way to do this is to connect to the database from the WebBroker application, and retrieve the data by running queries. You can also generate the xml by persisting ado recordsets to xml (produces very verbose xml!), manually coding it using the DOM.

The Sample Applications

The XSLT WebBroker Extensions can be downloaded from Code Central

The installation program for the WebBroker extensions that accompany this article installs the source for two example isapi dll's. Both use Microsoft Access as the database, however the choice of Access for the database was primarily to have a database format that most developers would be able to use. Not everyone uses Interbase or SQLServer. I typically use SQLServer for the apps that I develop but that is driven more customer demand rather than preference.

The first example uses the ADO Express components to connect to the Access Northwinds sample database. It shows how to use the components to display the customers table using the TADOQuery component, and makes use of the recordset paging facilities in ado recordsets.

The second example is more complex, it uses an MTS/COM+ object to return the data as disconnected ado recordsets, it also shows how to add more WebModules using the TXSLWebDispatcher component on Data Modules.

Reference:

The XSL/XSLT Specification: https://www.w3.org/Style/XSL/

XSLT Programmers Reference by Michael Kay, Wrox Press - ISBN: 1-861003-12-9

XSL School - Free web based tutorial: https://www.w3schools.com/xsl/

Enjoy! - Vincent Parrett, VSoft Technologies Pty Ltd, Canberra Australia https://www.vsoft-tech.com.au

Published on: 3/21/2001 6:08:40 PM

Server Response from: ETNASC01