Using XML and XSL in Pocket Internet Explorer

Get started in using XML, XSL, XML DOM, and JScript in Microsoft Internet Explorer for the Pocket PC as I walk you through some sample code.

Download 745-CF-DEV.exe.

Applies to:
- Your current Web application toolset
- XML support in Microsoft SQL Server 2000
- Microsoft Windows Powered Pocket PC 2000
- Microsoft Pocket Internet Explorer

Page Options
Average rating:
6 out of 9
Rate this page
Print this page
E-mail this page
Add to Favorites

Gotchas

When you create and edit XML (Extensible Markup Language) and XSL (Extensible Stylesheet Language) files, you have to do it very carefully. Neither XML nor XSL is "forgiving" when it comes to formatting and you might want to use a tool to help you find syntax errors.

Languages Supported

English

Separating Content From Layout

The whole idea with XML is to separate the content from the layout of a Web page. Traditionally, you have used HTML (HyperText Markup Language) to create your Web pages and if you have started to build server-side applications, you have created those HTML pages dynamically using something like ASP (Active Server Pages). As long as you only needed to support a limited number of clients (browsers), you could get away with using a "least denominator" approach (only include things that were supported by all clients/browsers).

As you may have seen in my article Make Your Web Applications Support Pocket PC, you can add support for multiple clients (I talked about Pocket PC and Microsoft Mobile Explorer in the article). You can do this by dynamically delivering different content depending on which type of client is accessing the ASP page using the HTTP (HyperText Transfer Protocol) information. That is a great way to get started on a multi-channel strategy, but if you want a more solid approach, you should look at XML.

The interesting thing is that nowadays Microsoft has added support for XML in most of its products and what is particularly interesting is that they have added support for XML and even XSL on the Pocket PC. XSL is primarily used to format XML into HTML. The XML/XSL support in Pocket Internet Explorer has many of the important features from the PC implementation.

XML Impact

One interesting aspect of separating content (data) from layout is that when the Web page needs to be updated, you only need to transfer the new data to the client. In a wireless scenario, where bandwidth is a serious issue, this is really interesting. Any amount of data that we can avoid passing "in the air" has a positive impact on cost.

Another issue is server load. If we do all the handling of content adapted for different clients, we have to do an awful lot of work on the server. If we would let the server simply provide new data on a "need-for-update" basis, we would not only save valuable server processing power, but also bandwidth at the same time.

Also, when we want to take Web pages offline another issue becomes important—storage space. If you copy all the Web pages of your application to a Pocket PC Web page cache, you will end up with all the layout (formatting) information stored over and over again (in each page).

A Catalog Sample

To show you how the impact for a business application, I have created a sample book catalog based on a table in the pubs database that comes with SQL Server 2000. When the user enters the address to the catalog XML file (titles.xml), this is how it looks:

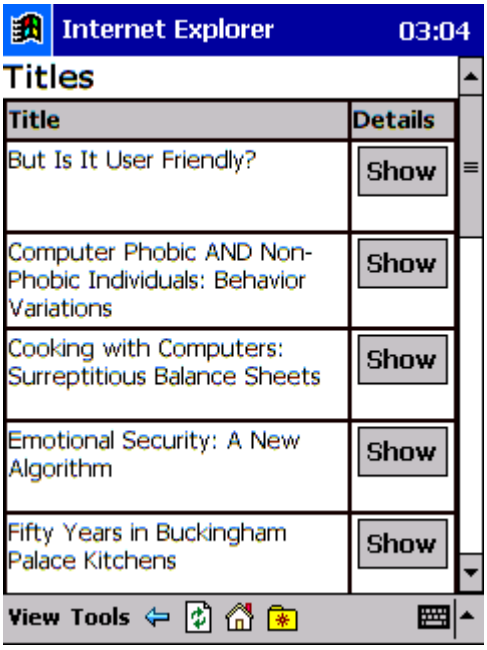


Figure 1: XML book catalog sample.

And when the user taps on the top **Show** button, this is what she gets:

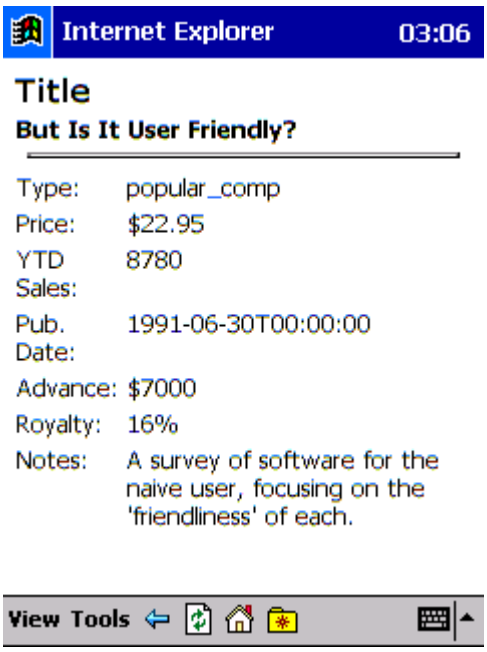


Figure 2: Book title details.

This is a fairly simple example of what you can do with XML and XSL and you might say that you could have done this with some server-side scripting as well. And you're right. The interesting thing about this sample is that both views (list and details) use the same source (in XML) for displaying the layout. If you had built the solution in ASP, you would have sent a huge amount of HTML over the wire (or in the air). Another interesting thing is what happens when the book information changes on the server. The field **YTD Sales** is probably something that changes often, and therefore you would probably like to update only the data, as the layout will not change as often.

Another interesting thing about the sample is that even if it's based on XML and XSL, you can take it offline. If you move the files from the server to a folder on your Pocket PC, you can view the same information without the need of any connection. If you would do the same thing for an ordinary Web application, you would probably end up with a lot of static Web pages in your offline Web page cache. In this simple sample, you end up with a demand for almost 20 times (!) as much space on your Pocket PC.

Sample Walk Through

When you start looking at the sample files, you can see that the XML source is a plain file containing all the titles (titles.xml). In a real-world scenario, this file would most probably have been generated from a database source. When I created the file, I used the XML support in SQL Server 2000. After setting up a virtual directory (pubs) for XML access to the **pubs** database, I

used the following URL (Universal Resource Locator) to get the XML data:

```
http://localhost/pubs?SQL=select+*+from+titles+for+xml+auto&root=titlelist
```

And if you would like to take a look yourself at out how the XML result is transformed using the XSL file (titlesSQL.xsl), you have to create a normal virtual directory (piexml) in Internet Information Server, place the XSL file (titlesSQL.xsl) there. Then try the following URL:

```
http://localhost/pubs?SQL=select+*+from+titles+for+xml+auto&root=titlelist&xsl=titlesSQL.xsl
```

Both the above URLs work in the PC version of Internet Explorer and Internet Explorer for the Pocket PC. As I have already exported the book data to a XML file (titles.xml), you will not need SQL Server 2000 to try the sample out.

Let's start by looking at part of the titles.xml file (first two book titles):

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml:stylesheet type="text/xsl" href="titles.xsl" ?>
<titlelist>
  <titles title_id="BU1032" title="The Busy Executive&apos;s Database Guide" type="business    " pub_id="1389
  price="19.99" advance="5000" royalty="10" ytd_sales="4095" notes="An overview of available database systems
  emphasis on common business applications. Illustrated." pubdate="1991-06-12T00:00:00"/>
  <titles title_id="BU1111" title="Cooking with Computers: Surreptitious Balance Sheets" type="business    "
  pub_id="1389" price="11.95" advance="5000" royalty="10" ytd_sales="3876" notes="Helpful hints on how to use
  electronic resources to the best advantage." pubdate="1991-06-09T00:00:00"/>
.
.
.
</titlelist>
```

You can see that each row in the database is stored as a **titles** XML tag and each column is an attribute on that tag. This may not be the most common way of formatting data in XML, but it is the native format that SQL Server 2000 uses. Even if you would store each column as a sub-tag to the row tag, the sample XSL files wouldn't look much different. On the second line you can see that the XML file points to a XSL file (titles.xsl) to use for formatting.

So, let's go on and look at the titles.xsl file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="uri:xsl">
  <xsl:template match="/">
    <SCRIPT>
    function showDetail(title_id)
    {
      var root = detailXSL.documentElement;
      var selectedElems =  root.selectNodes("//xsl:if");
      var ifStatement = selectedElems.item(0);
      ifStatement.attributes(0).text =
                     "@title_id[.='" + title_id + "']";

      document.write(titles.transformNode(detailXSL.documentElement));
    }
    </SCRIPT>
    <H3>Titles</H3>
    <TABLE BORDER="1" CELLPADDING="1" CELLSPACING="0">
      <TR>
        <TD BGCOLOR="#C0C0C0"><B>Title</B></TD>
        <TD BGCOLOR="#C0C0C0"><B>Details</B></TD>
      </TR>
      <xsl:for-each select="titlelist/titles" order-by="@title">
        <TR>
          <TD VALIGN="top"><xsl:value-of select="@title"/></TD>
          <TD VALIGN="top"><INPUT>
            <xsl:attribute name="TYPE">button</xsl:attribute>
            <xsl:attribute name="VALUE">Show</xsl:attribute>
            <xsl:attribute name="ONCLICK">showDetail('<xsl:value-of select="@title_id"/>')</xsl:attribute>
          </INPUT></TD>
        </TR>
      </xsl:for-each>
    </TABLE>
    <XML ID="titles" SRC="titles.xml"></XML>
    <XML ID="detailXSL" SRC="detail.xsl"></XML>
  </xsl:template>
</xsl:stylesheet>
```

If you just look at the HTML part for this file, you see that we first define a table with two columns (**Title** and **Details**) and then fill the table with a row for each title in the XML file. We also sort the title list on the title of the book. In the second column of

each row, we create a **Show** button that calls the JScript function (showDetail) declared in the top <SCRIPT> tag. This function loads the XSL file for viewing the details page (details.xsl defined in the XML "island" at the bottom), finds the "if" statement, and updates the title ID (title_id) to look for. The function finally transforms the XML file (titles.xml) using the XSL (details.xsl) and writes the output to the HTML document in the browser. The showDetail function uses the XML DOM (Document Object Model) to do its work.

We had better have a look at the details.xsl file too:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="uri:xsl">
  <xsl:template match="/">
    <H3>Title</H3>
    <TABLE BORDER="0" CELLPADDING="1" CELLSPACING="0">
      <xsl:for-each select="titlelist/titles">
        <xsl:if test="@title_id[.='??????']">
          <TR>
            <TD VALIGN="top" COLSPAN="2"><B><xsl:value-of
              select="@title"/></B><BR/><HR/></TD>
          </TR>
          <TR>
            <TD VAlIGN="top">Type:</TD>
            <TD VALIGN="top"><xsl:value-of select="@type"/></TD>
          </TR>
          <TR>
            <TD VAlIGN="top">Price:</TD>
            <TD VALIGN="top">$<xsl:value-of select="@price"/></TD>
          </TR>
          <TR>
            <TD VAlIGN="top">YTD Sales:</TD>
            <TD VALIGN="top"><xsl:value-of select="@ytd_sales"/></TD>
          </TR>
          <TR>
            <TD VAlIGN="top">Pub. Date:</TD>
            <TD VALIGN="top"><xsl:value-of select="@pubdate"/></TD>
          </TR>
          <TR>
            <TD VAlIGN="top">Advance:</TD>
            <TD VALIGN="top">$<xsl:value-of select="@advance"/></TD>
          </TR>
          <TR>
            <TD VAlIGN="top">Royalty:</TD>
            <TD VALIGN="top"><xsl:value-of select="@royalty"/>%</TD>
          </TR>
          <TR>
            <TD VAlIGN="top">Notes:</TD>
            <TD VALIGN="top"><xsl:value-of select="@notes"/></TD>
          </TR>
        </xsl:if>
      </xsl:for-each>
    </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

At the top of the file, you find the "if" statement (xsl:if) that was manipulated from the showDetail function in the titles.xsl file. And we only show the column values for the selected title in the resulting HTML page.

Conclusion

Hopefully, now you can fully agree with me that there are a number of reasons why you should start looking at XML and XSL for your multi-channel applications. You can save time in designing and implementing your Web applications, and you will also save server load and (wireless) network bandwidth. When taking your XML applications offline, you will also save storage space in your Pocket PC. My advice to you is to start diving into the world of XML and start separating content from layout in your Pocket PC Web applications—right now!

Print E-Mail Add to Favorites

How would you rate the quality of this content?

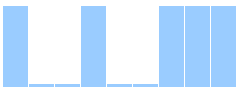
1 2 3 4 5 6 7 8 9

Poor Outstanding

Tell us why you rated the content this way. (optional)

Submit

Average rating:
6 out of 9



1 2 3 4 5 6 7 8 9

5 people have rated this page

Manage Your Profile | Legal | Contact Us | MSDN Flash Newsletter

© 2005 Microsoft Corporation. All rights reserved. Terms of Use | Trademarks | Privacy Statement