

## Office Talk

XML and Microsoft Office

---

Paul Cornell  
Microsoft Corporation

August 2, 2001

The *Extensible Markup Language* (XML) is a set of technologies that provides you with a platform-neutral and application-neutral format for describing data. This allows you to import and work with data that originates from applications outside of Microsoft® Office, as well as export data from Office to a myriad of other data formats that your business partners may require. XML will also comprise a large part of the Microsoft applications, operating systems, and technologies in the future, so learning about XML now will reap huge dividends for you as an Office developer.

If you've ever worked with Hypertext Markup Language (HTML), you should be able to work with XML. XML is similar to HTML in several ways:

- XML and HTML use similar characters (like < and >) to tag data.
- The tags that surround data (such as <B>, <OL>, <ADDRESS>, or <EMPLOYEEENUMBER>) give you clues as to what the information represents.
- XML and HTML can use Cascading Style Sheets (CSS) to display data in a variety of formats.

XML and HTML do have several differences, however:

- XML provides no inherit presentation capabilities, while Web browsers can display with reasonable fidelity any HTML that contains no additional presentation instructions.
- XML allows you to define an unlimited number of tags, while HTML is limited to a predefined group of tags.
- The Extensible Stylesheet Language (XSL) allows you to transform XML into a variety of different output formats, such as HTML or other dialects of XML. You can use XSL to transform HTML to other formats, but the HTML must conform to the XML syntax rules.

The XML syntax rules are fairly straightforward. Most of the rules can be summed up in the following list:

- Each XML document must have exactly one top-level element (also known as the *document element* or the *root element*). All of the other elements in the document must be nested within this element.
- All of the elements in the XML document must be properly nested. For example, <b><i>Text</i></b> is valid, but <b><i>Text</b></i> is not.
- Each element in the XML document must have both a starting tag and an ending tag. For example, <p>Text</p> is valid, but <p>Text is not.
- The name of the element in a starting tag must exactly match the name of the element in the corresponding ending tag, and element names are case sensitive. For example, <p>Text</p> is valid, but <P>Text</p> is not.
- Empty elements take the form <element></element> or just <element />. For example, <p></p> or <p /> is valid, but an empty <p> by itself is not.
- Certain characters are not allowed in certain locations in an XML document and must be replaced by *entities*. For example, instead of using the & character, you must use the entity &amp; instead. This is called *escaping* the character.
- If your XML document relies on a *Document Type Definition* (DTD) or an *XML Schema Definition* (XSD) file, your XML document must also conform to the rules set forth in the DTD or XSD file.

To demonstrate, here is a very minimal XML file:

```
<?xml version="1.0"?>
<me>
  <name role="author">Paul Cornell</name>
</me>
```

In this simple example:

- The code <?xml version="1.0"?> is a *processing instruction* (as noted by the symbols <? and ?>) that indicates to the XML *parser* (such as the one included with Microsoft Internet Explorer, also known as the *MSXML* parser) that this data is to be treated as XML. At the time that this column was written, the only valid value for the **version** attribute is 1.0.
- The code <me></me> defines the root element. All of the other elements must be contained within this root element.
- The **name** element above contains one *attribute*, **role**, with the value of **author**. The *content* of the **name** element is, of course, my name.

A complete discussion about XML in general is beyond the scope of this column. For more information about XML in general, including references and tutorials, [go to the MSDN Online XML Developer Center](#).

The rest of this month's column will focus on how to use XML in Office.

## Using XML in Your Office Solutions

In Microsoft Office 2000, there is very limited XML support. In fact, the only place that XML is used in Office 2000 is through the use of XML *data islands* in Microsoft Excel 2000, Microsoft PowerPoint® 2000, and Microsoft Word 2000 when these documents are saved as Web pages. This allows these documents to be viewed and edited in Microsoft Internet Explorer while maintaining the rich formatting that was used when these documents were created in their original applications.

In Microsoft Office XP, XML is implemented in several additional ways:

- You can save Microsoft Excel 2002 spreadsheets and Microsoft Access 2002 database tables, queries, and views as XML.
- You can import XML into Excel 2002 spreadsheets and Access 2002 databases.
- Microsoft Outlook® 2002 views are defined in XML. You can modify these view formats by using Visual Basic® for Applications (VBA) code.
- *Smart tags* can be embedded as XML inside of Microsoft Word 2002 documents, Excel 2002 spreadsheets, Outlook 2002 e-mail (when Word 2002 is enabled as your e-mail editor), or Web pages (when Office XP or one of the individual applications just mentioned) is installed on your computer. Reusable smart tags can also be written in XML and distributed to multiple Office XP users.

In the remainder of this column, I will demonstrate how to import and export XML to and from Access 2002 and Excel 2002. I will also show you how to modify an Outlook 2002 view using XML.

## Importing XML into Access

Let's first import some sample data into Access 2002. Let's say I have the following XML data file that represents a list of employees:

```
<?xml version="1.0" ?>
<Employees>
  <Employee>
    <Name>
      <FirstName>Scott</FirstName>
      <LastName>Cooper</LastName>
    </Name>
    <Address>
      <Street>123 Main St.</Street>
      <City>Anytown</City>
      <State>WA</State>
      <ZIP>99999</ZIP>
    </Address>
    <PhoneNumber>(555)555-1212</PhoneNumber>
    <HireDate>11-03-1997</HireDate>
  </Employee>
  <Employee>
    <Name>
      <FirstName>Katie</FirstName>
      <LastName>Jordan</LastName>
    </Name>
    <Address>
      <Street>234 Microsoft Way</Street>
      <City>Redmond</City>
      <State>WA</State>
      <ZIP>99199</ZIP>
    </Address>
    <PhoneNumber>(555)555-1234</PhoneNumber>
    <HireDate>03-10-1999</HireDate>
  </Employee>
  <Employee>
    <Name>
      <FirstName>William</FirstName>
      <LastName>Vong</LastName>
    </Name>
    <Address>
      <Street>543 - 16th Ave.</Street>
      <City>Lynnwood</City>
      <State>WA</State>
      <ZIP>99909</ZIP>
    </Address>
    <PhoneNumber>(555)555-2345</PhoneNumber>
    <HireDate>04-06-2000</HireDate>
  </Employee>
</Employees>
```

As you can see from the code above:

- The root element for this XML file is **Employees**. The **Employees** element contains three child **Employee** elements.
- Each **Employee** element contains four child elements: **Name**, **Address**, **PhoneNumber**, and **HireDate**.
- Likewise, each **Name** element contains two child elements: **FirstName** and **LastName**.
- Each **Address** element contains four child elements of its own: **Street**, **City**, **State**, and **ZIP**.

Let's import this table into Access 2002 and see what happens. To do this:

1. Using Notepad, save the XML code above to a text file named EmployeeList.xml.
2. Create a new blank Access 2002 database.
3. On the **File** menu, point to **Get External Data**, and then click **Import**.
4. In the **Import** dialog box, in the **Files of type** list, click **XML Documents (\*.xml, \*.xsd)**.
5. Locate and click the EmployeeList.xml file, click **Import**, and then click **OK**.

You will notice that three separate, unrelated tables were created as follows:

- The Address table contains the Street, City, State, and ZIP fields.
- The Employee table contains the PhoneNumber and HireDate fields.
- The Name table includes the FirstName and LastName fields.

Access made its best guess at importing the data, but that's probably not what we wanted. To create one table when we import the XML file into Access, we must provide the XML data to Access in this format:

```
<?xml version="1.0" ?>
<Employees>
  <Employee>
    <FirstName>Scott</FirstName>
    <LastName>Cooper</LastName>
    <Street>123 Main St.</Street>
    <City>Anytown</City>
    <State>WA</State>
    <ZIP>99999</ZIP>
    <PhoneNumber>(555)555-1212</PhoneNumber>
    <HireDate>11-03-1997</HireDate>
  </Employee>
  <Employee>
    <FirstName>Katie</FirstName>
    <LastName>Jordan</LastName>
    <Street>234 Microsoft Way</Street>
    <City>Redmond</City>
    <State>WA</State>
    <ZIP>99199</ZIP>
    <PhoneNumber>(555)555-1234</PhoneNumber>
    <HireDate>03-10-1999</HireDate>
  </Employee>
  <Employee>
    <FirstName>William</FirstName>
    <LastName>Vong</LastName>
    <Street>543 - 16th Ave.</Street>
    <City>Lynnwood</City>
    <State>WA</State>
    <ZIP>99909</ZIP>
    <PhoneNumber>(555)555-2345</PhoneNumber>
    <HireDate>04-06-2000</HireDate>
  </Employee>
</Employees>
```

To see how this works, using Notepad, copy the XML code above to a text file and save this file with the name FlattenedEmployees.xml. Then follow the previous steps to import the FlattenedEmployees.xml file into Access 2002 and notice that only one table is imported.

Short of retyping the XML file, how do we transform the data from one XML structure to another? The answer is by using an XML technology called *XSL Transformations* (XSLT), which is itself based on an XML technology called the *Extensible Stylesheet Language* (XSL). The MSXML parser can apply the instructions in an XSLT file to display XML in a different format. The MSXML parser also exposes an object model called the *XML Document Object Model* (DOM) that allows you to access the various elements in an XML file (among other things).

Here's what the XSLT instructions would look like to "flatten" our multidimensional XML file:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes" />
```

```

<xsl:template match="/">
  <Employees>
    <xsl:apply-templates select="//Employee" />
  </Employees>
</xsl:template>

<xsl:template match="Employee">
  <Employee>
    <xsl:apply-templates select="Name" />
    <xsl:apply-templates select="Address" />
    <xsl:apply-templates select="PhoneNumber" />
    <xsl:apply-templates select="HireDate" />
  </Employee>
</xsl:template>

<xsl:template match="Name">
  <xsl:apply-templates select="FirstName" />
  <xsl:apply-templates select="LastName" />
</xsl:template>

<xsl:template match="FirstName">
  <FirstName><xsl:value-of select="." /></FirstName>
</xsl:template>

<xsl:template match="LastName">
  <LastName><xsl:value-of select="." /></LastName>
</xsl:template>

<xsl:template match="Address">
  <xsl:apply-templates select="Street" />
  <xsl:apply-templates select="City" />
  <xsl:apply-templates select="State" />
  <xsl:apply-templates select="ZIP" />
</xsl:template>

<xsl:template match="Street">
  <Street><xsl:value-of select="." /></Street>
</xsl:template>

<xsl:template match="City">
  <City><xsl:value-of select="." /></City>
</xsl:template>

<xsl:template match="State">
  <State><xsl:value-of select="." /></State>
</xsl:template>

<xsl:template match="ZIP">
  <ZIP><xsl:value-of select="." /></ZIP>
</xsl:template>

<xsl:template match="PhoneNumber">
  <PhoneNumber><xsl:value-of select="." /></PhoneNumber>
</xsl:template>

<xsl:template match="HireDate">
  <HireDate><xsl:value-of select="." /></HireDate>
</xsl:template>

</xsl:stylesheet>

```

To try the transformation out yourself, using Notepad, save the code above to a file named `FlattenedEmployees.xsl`. To apply the `FlattenedEmployees.xsl` transformation to the `EmployeeList.xml` file, download [Msxsl.exe Command Line Transformation Utility](#).

The usage of `MSXSL.EXE` is as follows:

```
MSXSL.EXE source stylesheet -o output
```

Where `source` is the path and file name of the source XML file, `stylesheet` is the name of the XSLT stylesheet, and `output` is the path and file name of the transformed file to be output.

So to transform our XML above, we could use the following syntax:

```
MSXSL.EXE C:\My XML\EmployeeList.xml
C:\My XML\FlattenedEmployees.xsl -o C:\My XML\FlattenedEmployees.xml
```

While you could just add the following line of code to the `EmployeeList.xml` file, immediately after the `<?`

`xml version="1.0"?` processing instruction, and then view the `EmployeeList.xml` file in Microsoft Internet Explorer 5 or later:

```
<?xml-stylesheet type="text/xsl" href="FlattenedEmployees.xsl"?
```

All you would see is this:

```
Scott Cooper 123 Main St. Anytown WA 99999 (555)555-1212 11-03-1997
Katie Jordan 234 Microsoft Way Redmond WA 99199 (555)555-1234 03-10-1999
William Vong 543 - 16th Ave. Lynnwood WA 99909 (555)555-2345 04-06-2000
```

The reason for this is that Internet Explorer applies a *default template* to the transformed XML data. Because the transformed XML sits in memory and is not output to a file, Internet Explorer makes its best guess at what the XML should look like. In this case, Internet Explorer outputs a steady stream of text with no formatting.

## Exporting XML from Access

Now let's try exporting a table from Access. Let's use a simple table for this example—the `Categories` table in the Northwind sample database. To export this data:

1. In the Database Window in the Northwind sample database, right-click the `Categories` table, and then click **Export**.
2. In the **Save as type** list in the **Export Table 'Categories' To** dialog box, click **XML Documents (\*.xml)**.
3. In the **File name** list, type `Categories.xml`. Then click **Export**.
4. In the **Select what information will be exported** area of the **Export XML** dialog box, there are three check boxes:
  - **Data (XML)**: Checking this box also checks the **Export Data** box, selects the **Static Data** option, synchronizes the **Export Location** box to match the choices that were made in the **Export Table 'Categories' To** dialog box, and selects **UTF-8** as the encoding mechanism, on the **Data** tab (**Advanced** button). On the **Data** tab, you can change the export location, change the encoding mechanism, and even stop exporting the XML data.
  - **Schema of the data**: Checking this box also checks the **Export Schema** box and checks the **Include primary key and index information** box, selects the **Create separate schema document** option, and suggests a schema file name and location based on the name provided in the **Export Table 'Categories' To** dialog box, on the **Schema** tab (**Advanced** button). On the **Schema** tab, you can exclude the primary key and index information, embed the schema in the exported XML data, change the schema export location, and even stop exporting the schema.
  - **Presentation of your data (XSL)**: Checking this box also checks the **Export Presentation (HTML 4.0 Sample XSL)** box, selects the **Client (HTML)** option, and suggests a presentation file name and location based on the name provided in the **Export Table 'Categories' To** dialog box, on the **Presentation** tab (**Advanced** button). On the **Presentation** tab, you can run the presentation file from the server, include or exclude report images, change the presentation export location, and even stop exporting presentation details.

Let's examine what happens when I check all three boxes and accept the defaults.

## Examining the XML Data File and Its Syntax

Here is the `Categories.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?'>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata" xmlns:
xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Categories.xsd">
  <Categories>
    <CategoryID>1</CategoryID>
    <CategoryName>Beverages</CategoryName>
    <Description>Soft drinks, coffees, teas, beer, and
      ale</Description>
  </Categories>
  <Categories>
    <CategoryID>2</CategoryID>
    <CategoryName>Condiments</CategoryName>
    <Description>Sweet and savory sauces, relishes, spreads,
      and seasonings</Description>
  </Categories>
  <!-- Repeated information omitted for brevity. -->
  <Categories>
    <CategoryID>8</CategoryID>
    <CategoryName>Seafood</CategoryName>
    <Description>Seaweed and fish</Description>
  </Categories>
</dataroot>
```

As you can see, the XML data output is straightforward—the root element **dataroot** contains one child node for each row of the Categories table, and each **Categories** child node contains one child node for each column in the Categories table (**CategoryID**, **CategoryName**, and **Description**).

## Examining the XML Schema File and Its Syntax

Here is the Categories.xsd file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:od="urn:schemas-microsoft-com:officedata">
  <xsd:element name="dataroot">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element ref="Categories"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Categories">
    <xsd:annotation>
      <xsd:appinfo>
        <od:index index-name="CategoryName"
          index-key="CategoryName"
          primary="no" unique="yes" clustered="no"/>
        <od:index index-name="PrimaryKey" index-key="CategoryID"
          primary="yes" unique="yes" clustered="no"/>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CategoryID" od:jetType="autonumber"
          od:sqlType="int" od:autoUnique="yes"
          od:nullable="yes">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer"/>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="CategoryName" minOccurs="0"
          od:jetType="text"
          od:sqlType="nvarchar">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:maxLength value="15"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="Description" minOccurs="0"
          od:jetType="memo"
          od:sqlType="ntext">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:maxLength value="536870910"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

This schema file is a little bit longer than the XML file, but it's readable if you dig into it. The root element **schema** contains one **element** element for each element in the XML file (**dataroot**, **Categories**, **CategoryID**, **CategoryName**, and **Description**). The **dataroot** element states that it can contain an unlimited number of **Categories** elements. The **Categories** element states that it must contain a **CategoryID** element of type **Integer** that is automatically numbered, may contain a **CategoryName** element with up to 15 characters, and may contain a **Description** element with up to 536,870,910 characters. Note that without this XSD file, all of the data would be treated as ordinary text.

## Examining the XSL Transformation File and Its Syntax

Here is the Categories.xsl file:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  language="vbscript">
  <xsl:template match="/">
    <HTML>
```

```

<HEAD>
  <META HTTP-EQUIV="Content-Type"
    CONTENT="text/html; charset=UTF-8" />
  <TITLE>Categories</TITLE>
  <STYLE TYPE="text/css" />
</HEAD>
<BODY link="#0000ff" vlink="#800080">
  <TABLE BORDER="1" BGCOLOR="#ffffff" CELSPACING="0" CELLPADDING="0">
    <TBODY>
      <xsl:for-each select="/dataroot/Categories">
        <xsl:eval>AppendNodeIndex(me)</xsl:eval>
      </xsl:for-each>
      <xsl:for-each select="/dataroot/Categories">
        <xsl:eval>CacheCurrentNode(me)</xsl:eval>
        <xsl:if expr="OnFirstNode">
          <TR>
            <TH style="width: 0.875in">Category ID</TH>
            <TH style="width: 1.125in">Category Name</TH>
            <TH style="width: 3.7916in">
              Description</TH>
          </TR>
        </xsl:if>
        <TR>
          <TD>
            <xsl:eval no-entities="true">
              Format(GetValue("CategoryID", 3),"", "")
            </xsl:eval>
          </TD>
          <TD>
            <xsl:eval no-entities="true">
              Format(GetValue("CategoryName", 202),"", "")
            </xsl:eval>
          </TD>
          <TD>
            <xsl:eval no-entities="true">
              Format(GetValue("Description", 202),"", "")
            </xsl:eval>
          </TD>
        </TR>
        <xsl:if expr="OnLastNode" />
        <xsl:eval>NextNode()</xsl:eval>
      </xsl:for-each>
    </TBODY>
  </TABLE>
</BODY>
</HTML>
<xsl:script>
  <![CDATA[
    <!-- Script omitted for brevity. -->
  ]]>
</xsl:script>
</xsl:template>
</xsl:stylesheet>

```

This file is actually quite large, so I've omitted most of it for brevity. The **stylesheet** element contains one **template** element that in turn contains all of the code to transform the Categories.xml file into a reasonable facsimile of an Access database table.

## Importing XML into Excel

Now that we've looked at Access, let's see how you can import and export XML to and from Excel 2002. First, let's look at importing data. For this example, we'll use the list of employees that we used for the Access example (EmployeeList.xml). If we open the EmployeeList.xml file in Excel (by clicking **Open** from the **File** menu), we see the following (the spreadsheet has been cropped for space):



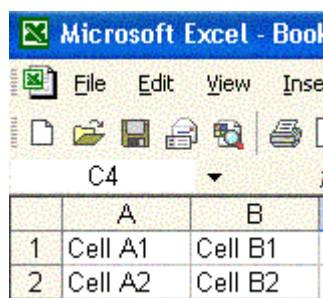
/Employees			
/Employee/Address/City	/Employee/Address/State	/Employee/Address/Street	/Employee/Address
Anytown	WA	123 Main St.	!
Redmond	WA	234 Microsoft Way	!
Lynnwood	WA	543 - 16th Ave.	!
/Employee/Address/ZIP/#agg			
/Employee/HireDate		/Employee/Name/FirstName	
99999		11-03-1997	Scott
99199		03-10-1999	Katie
99909		04-06-2000	William
			Cooper
			Jordan
			Vong

**Figure 1. The EmployeeList.xml file opened in Excel**

On first glance, this doesn't look much like the EmployeeList.xml file. This is because our XML file is multidimensional, but Excel uses a two-dimensional paradigm. To make the XML data fit into the Excel two-dimensional row-and-column format, Excel invokes a *spreadsheet flattener*. If you study Figure 1, you'll notice that the column headings in gray correspond to the elements in the XML file. Excel lists the columns in alphabetical order. To make the XML data appear differently in the spreadsheet, you would need to apply an XSLT file to the XML data. To make the XML data look more like a full-fidelity Excel spreadsheet, the XSLT file would need to include transformation instructions that adhere to the XML Spreadsheet Schema (XMLSS). The XMLSS documentation is available at [the Office Developer Center](#). To get a feel for what the XMLSS looks like, let's export an Excel spreadsheet as XML.

## Exporting XML from Excel

Exporting an Excel spreadsheet to XML is simple. To try this out, type the following values in an Excel spreadsheet, as shown in Figure 2.



**Figure 2. The sample Excel spreadsheet**

Then, by clicking **Save As** on the **File** menu, and clicking **XML Spreadsheet (\*.xml)** in the **Save as type** list on the **Save As** dialog box, you get an XML file that adheres to the XMLSS. Here's what the XML representation of **Figure 2** looks like:

```
<?xml version="1.0"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:html="http://www.w3.org/TR/REC-html40">
  <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
    <Author>Paul Cornell</Author>
    <LastAuthor>Paul Cornell</LastAuthor>
    <Created>2001-08-02T16:08:21Z</Created>
    <Company>Microsoft Corporation</Company>
    <Version>10.2625</Version>
  </DocumentProperties>
  <OfficeDocumentSettings xmlns="urn:schemas-microsoft-com:office:office">
    <DownloadComponents/>
    <LocationOfComponents HRef="file:///\\OfficeXP\\CD1\\"/>
  </OfficeDocumentSettings>
  <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
    <WindowHeight>6135</WindowHeight>
    <WindowWidth>8445</WindowWidth>
    <WindowTopX>240</WindowTopX>
    <WindowTopY>120</WindowTopY>
    <ProtectStructure>False</ProtectStructure>
    <ProtectWindows>False</ProtectWindows>
  </ExcelWorkbook>
  <Styles>
    <Style ss:ID="Default" ss:Name="Normal">
```



```

        <Alignment ss:Vertical="Bottom"/>
        <Borders/>
        <Font/>
        <Interior/>
        <NumberFormat/>
        <Protection/>
    </Style>
</Styles>
<Worksheet ss:Name="Sheet1">
    <Table ss:ExpandedColumnCount="2" ss:ExpandedRowCount="2"
        x:FullColumns="1" x:FullRows="1">
        <Row>
            <Cell><Data ss:Type="String">Cell A1</Data></Cell>
            <Cell><Data ss:Type="String">Cell B1</Data></Cell>
        </Row>
        <Row>
            <Cell><Data ss:Type="String">Cell A2</Data></Cell>
            <Cell><Data ss:Type="String">Cell B2</Data></Cell>
        </Row>
    </Table>
    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
        <Selected/>
        <Panes>
            <Pane>
                <Number>3</Number>
                <ActiveRow>3</ActiveRow>
                <ActiveCol>2</ActiveCol>
            </Pane>
        </Panes>
        <ProtectObjects>False</ProtectObjects>
        <ProtectScenarios>False</ProtectScenarios>
    </WorksheetOptions>
</Worksheet>
<Worksheet ss:Name="Sheet2">
    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
        <ProtectObjects>False</ProtectObjects>
        <ProtectScenarios>False</ProtectScenarios>
    </WorksheetOptions>
</Worksheet>
<Worksheet ss:Name="Sheet3">
    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
        <ProtectObjects>False</ProtectObjects>
        <ProtectScenarios>False</ProtectScenarios>
    </WorksheetOptions>
</Worksheet>
</Workbook>

```

The XML is straightforward—a single **Workbook** element contains a **DocumentProperties** element, an **OfficeDocumentSettings** element, an **ExcelWorkbook** element, and a **Styles** element that describes the workbook's properties, settings, and styles. This is followed by three **Worksheet** elements—one element for each standard worksheet in a new Excel workbook. The first **Worksheet** element contains **Table**, **Row**, and **Cell** elements that list the values of cells A1 through B2, as well as additional elements for property settings. Because worksheets two and three are empty, the rest of the elements don't have anything interesting to offer for this discussion.

If you open this XML file in Excel, it will look exactly like Figure 2.

## Working with Outlook Views in XML

Let me round out this month's column by showing you how to manipulate Outlook views through XML. In case you're not familiar with what a view is, a view in Outlook is the on-screen representation of Outlook data, such as your Inbox or your Calendar. Within the Calendar, you can view data in Day/Week/Month view, Recurring Appointments view, and so on. Working with Outlook views in XML is interesting, because, just like the previous section, you can create and manipulate full-fidelity Outlook views by using a simple text editor (like Notepad). And, similar to XMLSS, you use the Outlook XML View Definitions to specify the XML in such a way that Outlook can properly parse and display the view representation. The Outlook XML View Definitions are available in Morley Tooke's article [XML View Definitions](#).

To retrieve or change an Outlook view through XML, you need to use Microsoft Visual Basic for Applications (VBA) code along with the Outlook XML View Definitions for the view. Here's an example of how to use Outlook VBA and XML to create a new Calendar view for users with limited vision. First, here's the Outlook VBA code:

```

Sub CreateLargePrintCalendarView()

    Dim olApplication As Application
    Dim objNameSpace As NameSpace

```

```

Dim objViews As Views
Dim objNewView As View

' Set a reference to the Microsoft Scripting Runtime first.
Dim objFSO As FileSystemObject
Dim objTextStream As TextStream
Dim strXML As String

' Add a new Calendar view.
Set olApplication = Application
Set objNameSpace = olApplication.GetNamespace(Type:="MAPI")
Set objViews = objNameSpace.GetDefaultFolder(FolderType:=olFolderCalendar).Views
Set objNewView = objViews.Add(Name:="Large Print Calendar View", _
    ViewType:=olCalendarView, _
    SaveOption:=olViewSaveOptionThisFolderOnlyMe)

' Open the XML representation of the view.
Set objFSO = New FileSystemObject
Set objTextStream = objFSO.OpenTextFile
(File Name:="C:\My XML\ViewDef.xml", _
    IOMode:=ForReading)

' Use the XML in the ViewDef.xml file to define the appearance of the view.
strXML = objTextStream.ReadAll
objNewView.XML = strXML

End Sub

```

The VBA code above creates a new Calendar view. The code uses the XML in the ViewDef.xml file (an XML file that I created) to increase the view's font size. Here's the ViewDef.xml file:

```

<?xml version="1.0"?>
<view type="calendar">
  <viewname>New Calendar View</viewname>
  <viewstyle>table-layout:fixed;width:100%;font-family:Tahoma;font-style:normal;
font-weight:normal;font-size:8pt;color:Black</viewstyle>
  <viewtime>210596758</viewtime>
  <mode>0</mode>
  <splitterwidth>150</splitterwidth>
  <splitterheight>128</splitterheight>
  <displaytimeunits>30</displaytimeunits>
  <taskfilter>1</taskfilter>
  <includenodue>1</includenodue>
  <lunarinfo type>0</lunarinfo type>
  <rokuyou>0</rokuyou>
  <startfield>urn:schemas:calendar:dtstart</startfield>
  <endfield>urn:schemas:calendar:dtend</endfield>
  <daystyle>font-size:12pt</daystyle>
  <weekstyle>font-size:12pt</weekstyle>
  <monthstyle>font-size:12pt</monthstyle>
  <calendarstyle>font-size:12pt</calendarstyle>
  <previewpane>
    <markasread>0</markasread>
    <alignbottom>0</alignbottom>
    <previewwidth>0</previewwidth>
    <previewheight>500</previewheight>
  </previewpane>
  <embeddedview type="table">
    <viewstyle>table-layout:fixed;width:100%;font-family:Tahoma;font-style:normal;
font-weight:normal;font-size:8pt;color:Black</viewstyle>
    <viewtime>210596758</viewtime>
    <linecolor>8421504</linecolor>
    <linestyle>3</linestyle>
    <gridlines>1</gridlines>
    <newitemrow>1</newitemrow>
    <incelledit>1</incelledit>
    <collapsestate></collapsestate>
    <rowstyle>background-color:#ffffff</rowstyle>
    <headerstyle>background-color:#d3d3d3</headerstyle>
    <previewstyle>color:Blue</previewstyle>
    <column>
      <name>HREF</name>
      <prop>DAV:href</prop>
      <checkbox>1</checkbox>
    </column>
    <column>
      <heading>Icon</heading>
      <prop>http://schemas.microsoft.com/mapi/proptag/0x0fff0102</prop>

```

```

        <bitmap>1</bitmap>
        <maxrows>6619136</maxrows>
        <width>18</width>
        <style>padding-left:3px;text-align:center</style>
    </column>
    <column>
        <format>boolicon</format>
        <heading>Complete</heading>
        <prop>http://schemas.microsoft.com/mapi/id/
        {00062003-0000-0000-C000-000000000046}/811c000b</prop>
        <type>boolean</type>
        <bitmap>1</bitmap>
        <maxrows>7471104</maxrows>
        <width>35</width>
        <style>padding-left:3px;text-align:center</style>
        <displayformat>3</displayformat>
    </column>
    <column>
        <heading>TaskPad</heading>
        <prop>urn:schemas:httpmail:subject</prop>
        <type>string</type>
        <maxrows>6619136</maxrows>
        <width>87</width>
        <style>padding-left:3px;text-align:left</style>
        <userheading>TaskPad</userheading>
    </column>
    <groupbydefault>0</groupbydefault>
    <previewpane>
        <markasread>0</markasread>
        <previewwidth>0</previewwidth>
        <previewheight>500</previewheight>
    </previewpane>
</embeddedview>
</view>

```

You can learn more about how to create Outlook views through XML by reading Morley Tooke's article [Creating Custom Views](#).

## Where to Get More Info

- For more information about XML in general, including references and tutorials, see the [MSDN Online XML Developer Center](#).
- Download [Msxsl.exe](#). It enables you to easily and quickly transform XML files by using XSLT style sheets.
- The Outlook XML View Definitions are available in Morley Tooke's article [XML View Definitions](#).
- You can learn more about how to create Outlook views through XML by reading Morley Tooke's article [Creating Custom Views](#).
- As always, check in regularly at [the Office Developer Center](#) for information and technical articles on Office solution development.

---

**Paul Cornell** works for the MSDN Online Office Developer Center and the Office developer documentation team. He spends his free time playing card games with his wife and reading to his daughter.

---

[Send feedback to Microsoft](#)

[© 2004 Microsoft Corporation. All rights reserved.](#)