.NET Framework Support for XML in the Database Curriculum

Charles M. (Mike) Morrison
Associate Professor
Department of Computer Science
University of WI – Eau Claire
Eau Claire, WI 54701
715 836-4425
morriscm@uwec.edu

ABSTRACT

The *eXtensible Markup Language (XML)* provides a way to share data among applications that use different data storage technologies. Teaching database oriented XML concepts is enhanced when students are assigned exercises illustrating how XML documents are created, transformed, and validated. This paper explains how Microsoft's .NET 2.x Framework classes can be used to work with XML.

1 INTRODUCTION

XML is a topic in most database courses and teaching database oriented XML concepts is enhanced when students are given exercises illustrating how XML documents are created, transformed, and validated.

Microsoft's .NET Framework supports the DOM and SAX models for interacting with XML documents. The .NET DOM model is implemented through the XmlDocument class. This requires the more overhead than SAX, but adds the capability of editing an XML document and traversing back and forth through the nodes in the document. The SAX model is implemented through the XmlTextReader and XmlTextWriter classes. These SAX classes provide a fast, non-cached, forward-only method for reading, creating and writing an XML document.

.NET datasets are objects used to store data retrieved from databases, XML documents, and other sources. A dataset's ReadXml method retrieves the data within an XML file and stores it in the dataset. Datasets can be bound to interface controls that display and allow editing the data. Any data stored in a dataset, including data retrieved from databases and other sources, can be written in XML format by calling the dataset's WriteXml method.

Validating an XML document with an associated XSD is done using a combination of the XmlReader and XmlReaderSettings classes. An XmlReaderSettings object is associated with the XSD file. The XmlReader object is associated with the SmlReaderSettings object. Each XML node is then validated at the time it is retrieved with the XmlReader.

.NET supplies an XSLT processor in its XslCompiledTransform class making XSLTs an option for transforming an XML document into the different formats that are needed by applications using different data storage schemas and technologies.

This paper explains how to use the XML classes Microsoft placed in its .NET 2.x Framework for reading, writing, transforming, and validating XML.

2 XML Overview

As with HTML, XML uses tags and attributes to define data. Unlike HTML, however, XML allows developers to create custom tags that define data items and relationships.

The original intent of XML was to markup content – XML combined with cascading style sheets can be used to create custom tags formatting text displayed within browsers. It quickly became clear, however, that XML-formatted text files, also called XML documents, were equally well suited for storing structured data in a text file. Since then, XML has become the common denominator for database applications that share data across multiple organizations, or that share data across different hardware and software platforms.

2.1 XML Document Structure

Individual data items within an XML document are called *elements*. Elements can have hierarchical relationships in which one parent element has multiple related child elements. Figure 1 shows an example of an XML document with a root element named allstudents containing two child elements named studentdata. Studentdata in turn contains child elements named lastname and firstname.

```
<allstudents>
   <studentdata>
      <lastname>Wallen</lastname>
      <firstname>Clifford</firstname>
   </studentdata>
   <studentdata>
      <lastname>Volovsek</lastname>
      <firstname>Marion</firstname>
   </studentdata>
</allstudents>
```

Figure 1. XML document

2.2 Extensible Stylesheet Language Transformations (XSLTs)

The eXtensible Stylesheet Language, XSL, provides programming commands and structures to process and format XML data. An XSL program is called an XSL stylesheet or an eXtensible Stylesheet Language Transformation, XSLT. XSLTs enable programmers to sort, filter, modify, and format XML data, as well as transform its data structure. Although any language can be used to transform XML, XSL was specifically designed for this purpose and can simplify and speed up the process of writing a program to transform XML from one format to another.

XSL is built on the XML Path Language, XPath, a W3C standard. XPath is used to locate nodes within an XML document and it also provides a programming language with functions for working with number, string, and Boolean data values.

XSL templates are used to retrieve XML data, while XPath provides the syntax for describing the path to a particular part of an XML document, provides a mix of basic programming language instructions, and provides a number of functions that can be used to describe and manipulate XML data. XPath paths are similar to directory and folder paths and are specified with expressions like /inventory/item/inventorysize.

If an application expects to see Figure 1's data formatted with a root node named **students**, child elements named **student**, and with student having a single child element of **studentname**, the XSLT example shown in Figure 3 will transform the Figure 1's XML into what is shown in Figure 4.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
   <students>
      <xsl:for-each select="AllStudents/StudentData">
         <student>
            <studentname>
               <xsl:value-of select="lastname"/>,
               <xsl:value-of select="firstname"/>
            </studentname>
         </student>
      </xsl:for-each>
   </students>
</xsl:template>
</xsl:stylesheet>
```
Figure 3. XSLT example

```
<students>
  <student>
    <studentname>Wallen, Clifford</studentname>
  </student>
  <student>
    <studentname>Volovsek, Marion</studentname>
  </student>
<student>
```
Figure 4. Transformed XML

2.3 XML Schema Definitions

Database tables have fields, field data types, field constraints, and integrity constraints for primary and foreign keys. XML documents, however, don't support data types and constraints. Something additional is needed if this information is to be retained. An XML Document Type Definition, DTD, specifies which pairs of XML tags are required, which are optional, and the order in which they should be nested. DTDs are incapable of specifying all the data types and constraints possible in a relational database, however. XML Schema Definitions, XSDs, were developed for this purpose and are of more interest to a database course.

An XML Schema Definition (XSD) defines the data's structure in terms of data types, relationships, order and grouping mechanisms, and constraints. To validate an XML document, an external program is used to compare an XML file to its corresponding XSD file. XSDs can be used to describe an existing relational database structure, and then *migrate*, or move, the database data to an XML document. The XSD stores all of the database structure information from the original database while the XML document stores the actual data. Figure 5 shows the part of a schema definition for the XML in Figure 1 specifying lastname and firstname are strings limited to 30 characters and that lastname is required and firstname is not required.

```
<xs:element name="lastname" maxOccurs="1">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:maxLength value="30" />
   </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="firstname" maxOccurs="1" minOccurs="0">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:maxLength value="30" />
   </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 5. XML Schema Definition

3 .NET Framework Support for XML

Classes within the .NET Framework support reading, writing, parsing, and transforming XML. Both the Document Object Model, DOM, and Simple API for XML, SAX, models for parsing are supported.

In addition, .NET datasets provide methods for generating XML and XSDs, XML Schema Definitions, from data stored within them. Therefore, any data that can be read into a dataset (from a database table, spreadsheet, or anywhere else) can quickly be converted to XML.

Microsoft's Visual Studio applications are usually written using the VB or C# development tools supplied with Visual Studio. Over twenty other languages can be used however, so long as the language's compiler generates Microsoft Intermediate Language, MSIL, which is a low level language designed to be used by the .NET Framework's common language runtime, CLR. When the CLR runs MSIL for the first time, it compiles it into binary code and caches the compiled binary file to allow faster startups when the program is run in the future.

The XML examples in this paper will be shown using VB, however they could have been written using C# or any other MSIL compatible language.

3.1 Reading XML Documents

When reading an existing XML document, one of the following .NET Framework 2.x classes is normally used:

- XmlTextReader – This is the basic parser in .NET. It is read only, forward only and is Microsoft's version of SAX.
- XmlDocument – This is Microsoft's DOM implementation. The XML document is read into memory and presented in a tree like fashion. Updates are allowed to the XML.

3.1.1 XmlTextReader

The XmlTextReader makes sense when you don't need to update an XML file and want the fastest possible execution time. Figure 6 shows the code needed to open and read through an XML file.

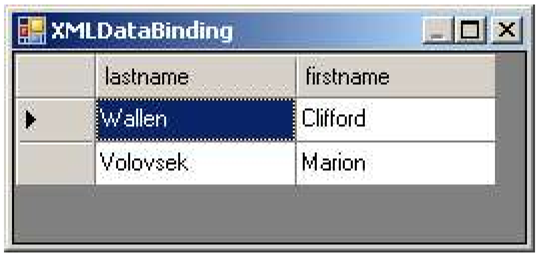
```
Dim tr As New System.Xml.XmlTextReader("student.xml")
Dim xmlOutput As String = ""
While tr.Read
   If tr.NodeType <> Xml.XmlNodeType.Whitespace And tr.Value <> "" Then
        xmlOutput += tr.Value & vbCr
   End If
End While
MessageBox.Show(xmlOutput)
```

Figure 6. Reading an XML file

An easier way to read an XML file takes advantage of a .NET dataset's ReadXml method. This method is derived from the XmlTextReader class (and the XmlTextReader's parent XmlReader class). Since datasets can be data bound to interface controls, a wide variety of choices are then available for displaying the file. For example the code in Figure 7 will read the XML from Figure 1 into a dataset and bind it to a DataGridView control. Figure 8 shows how the data is displayed in the DataGridView control.

```
Dim ds As New System.Data.DataSet
ds.ReadXml("Student.xml")
dgvStudent.DataSource = ds.Tables(0)
```

Figure 7. Reading an XML file into a dataset



	lastname	firstname
▶	Wallen	Clifford
	Volovsek	Marion

Figure 8. Displaying the XML in a DataGridView control

DataGridView controls can also be used to insert, update, and delete nodes from an XML document.

3.1.2 XmlDocument

The XmlDocument class provides support for DOM Level 2 and is possibly the most used XML class. Its extra overhead is often acceptable due to its navigation and updating capabilities. XmlDocument objects have methods for loading an XML file into memory, retrieving sets of nodes based on XPath locations and filters, traversing sets of nodes, appending new nodes, editing current nodes, and saving changes.

Figure 9 shows how to navigate through Figure 1's XML file using an XmlDocument object. The arguments .NET adds to event handlers have been replaced with ellipses (…) due space constraints. The Windows form containing this code is named xmlDocNavigate and there are textbox controls named txtLast and txtFirst on the form to display the contents of the nodes. A button control named btnNext is used to move from one node to the next.

An XmlNodeList is used to store the set of nodes retrieved by the XmlDocument object's SelectNodes method. Figure 1's studentdata element has two child elements, lastname and firstname. This knowledge is used in DisplayStudent when setting currentNode to the first child, assigning its InnerXml to txtLast, moving to the NextSibling, and then assigning its InnerXml to txtFirst. Since there are only two child nodes there's no need to call NextSibling again. Buttons for moving backwards, moving to the first node and to the last node aren't included in this example, but can be easily added using similar operations (with minor changes).

```
Public Class xmlDocNavigate
   Private xmlDoc As New System.Xml.XmlDocument
   Private currentNode As System.Xml.XmlNode
   Private nodeList As System.Xml.XmlNodeList
   Private currentIndex As Integer = 0

   Private Sub xmlDocNavigate_Load(…) Handles MyBase.Load
      xmlDoc.Load("student.xml")
      nodeList = xmlDoc.SelectNodes("allstudents/studentdata")
      DisplayStudent()
   End Sub

   Private Sub btnNext_Click(…) Handles btnNext.Click
      If currentIndex < nodeList.Count - 1 Then
         currentIndex += 1
         DisplayStudent()
      End If
   End Sub

   Private Sub DisplayStudent()
      currentNode = nodeList.Item(currentIndex).FirstChild
      txtLast.Text = currentNode.InnerXml
      currentNode = currentNode.NextSibling
      txtFirst.Text = currentNode.InnerXml
   End Sub
End Class
```
Figure 9. Navigating within an XmlDocument object

3.1.3 Validating XML from an XSD

Validating XML from an XSD requires creating an XmlReader object and an XmlReaderSettings object. The XmlReaderSettings object is associated with the XSD file. Then when the XmlReader object is instantiated, the XmlReader is associated with the XmlReaderSettings object (containing the XSD association). Each node is validated at the time it is retrieved with the XmlReader. Figure 10 shows how this is done. Note that if this seems different from what you might have done in the past using the 1.x .NET Framework, the newer 2.x .NET Framework classes are used in these examples.

```
Public Sub ValidateXML()
   Dim rs As New System.Xml.XmlReaderSettings
   Dim xr As System.Xml.XmlReader
   Try
      rs.Schemas.Add("", "student.xsd")
      rs.ValidationType = Xml.ValidationType.Schema
   Catch ex As Exception
      MessageBox.Show(ex.Message & vbCr & "Can't perform schema validation", _
                              "Error reading XSD file")
   End Try
   xr = Xml.XmlReader.Create("student.xml", rs)
   Try
      ' loop through all the nodes in the file to validate them
      While xr.Read()
         ' no additional code is needed here
      End While
   Catch ex As Exception
      MessageBox.Show("Validation Error: " & ex.Message)
   End Try
   xr.Close()
End Sub
```
Figure 10. Using an XSD to validate an XML document

This catches missing but required elements, oversize entries, incorrect data types, and any other schema definition errors specified in the XSD.

3.2 Writing XML Documents

SQL Server can retrieve data in various XML formats by adding the FOR XML clause to SELECT queries [1]. This can then be written to a file using any of .NET's file I/O methods.

.NET also provides ways to write XML from data that isn't necessarily formatted as XML when it is initially retrieved. Any data, once stored within a dataset, can be written as XML using a dataset's WriteXml method. Datasets also have a WriteXmlSchema method that will infer the schema definition for a set of data stored within it and write the corresponding XML to a file.

Figure 11 shows how to make a connection to a SQL Server 2005 Express database file and retrieve data into a dataset. (A modified ConnectionString property would be needed if connecting to a remote SQL Server instance.) The next to last line of code creates the XML file and the last line of code creates an XSD file from the structure of the data within the XML file.

```
Private Sub WriteXML()
   Dim cn As New Data.SqlClient.SqlConnection
   Dim da As New Data.SqlClient.SqlDataAdapter
   Dim sqlCmd As New Data.SqlClient.SqlCommand
   Dim ds As New Data.DataSet

   cn.ConnectionString = "Data Source=.\SQLEXPRESS;" & _
                         "AttachDbFilename=" & _
                         "|DataDirectory|\DatabaseName.mdf;" & _
                         "Integrated Security=True;"

   sqlCmd.CommandText = "SELECT * FROM UniversityStudent"

   da.SelectCommand = sqlCmd
   da.SelectCommand.Connection = cn

   da.Fill(ds, "StudentDataTable")

   ds.Tables("StudentDataTable").WriteXml("Student.xml")
   ds.Tables("StudentDataTable").WriteXmlSchema( "Student.xsd")
End Sub
```
Figure 11. Creating an XML file from data in a dataset

If you need finer control over how XML is written to a file, the XmlTextWriter and XmlDocument classes provide ways to do this. The XmlDocument class provides higher level, easier methods for writing XML than the XmlTextWriter class, but oddly, has no methods for creating an XML file if it doesn't already exist. Creating an application with a backend XML file for storing data entries is done with a combination of an XmlTextWrite object to create and initialize the XML file if it doesn't exist, and an XmlDocument object to insert, update, and delete entries in the file.

The code listing in Figure 9 can be modified to allow creating the file if it doesn't initially exist by adding the shaded code shown in Figure 12. Since an XmlDocument object can't create a file, an XmlTextWriter object is used.

```
Private Sub xmlDocNavigate_Load(…) Handles MyBase.Load
   Try
      xmlDoc.Load("student.xml")
   Catch ex As Exception
      ' if file isn't found, create it
      Dim tw As New System.Xml.XmlTextWriter("student.xml", _
                                                        System.Text.Encoding.UTF8)
      tw.WriteStartElement("allstudents")
      tw.Close()
      xmlDoc.Load("Courses.xml")
   End Try
   nodeList = xmlDoc.SelectNodes("allstudents/studentdata")
   DisplayStudent()
End Sub
```

Figure 12. Creating an XML document with XmlTextWriter

Appending new entries to the XML file requires an XmlDocument's CreateElement and AppendChild methods. Figure 13 demonstrates how these are used.

```
Private Sub btnSubmit_Click(…) Handles btnSubmit.Click
   Dim newNode As XmlElement = xmlDoc.CreateElement("studentdata")
   xmlRoot.AppendChild(newNode)

   Dim lastnameNode As XmlElement = xmlDoc.CreateElement("lastname")
   lastnameNode.InnerText = txtLast.Text
   newNode.AppendChild(lastnameNode)

   Dim firstnameNode As XmlElement = xmlDoc.CreateElement("firstname")
   firstnameNode.InnerText = txtFirst.Text
   newNode.AppendChild(firstnameNode)
End Sub
```

Figure 13. Appending to an XML file

The XmlDocument object will store changes and appended entries in memory. Writing the changes to a file can be done at whenever needed by using the XmlDocument's Save method:

xmlDoc.Save("filename.xml")

3.3 Transforming XML Documents

With a combination of XmlTextReader and XmlTextWriter a custom program can be written to read an XML file and write it back in format desired… An easier solution, however, is to write an XSLT and use that to transform the original XML into whatever is required. The .

NET XslCompiledTransform class contains Microsoft's XSLT processor. Figure 14 shows how an XSLT can be applied to an XML file with three lines of code.

```
Dim xslt As New System.Xml.Xsl.XslCompiledTransform
xslt.Load("student.xsl")
xslt.Transform("Student.xml", "StudentReformatted.xml")
```

Figure 13. Transforming an XML file

4 SUMMARY

Microsoft's .NET Framework supports the DOM model for interacting with XML documents with the XmlDocument class. This requires the more overhead than the SAX classes, but allows editing an XML document and traversing back and forth through the nodes in the document.

The SAX model is supported by the XmlTextReader and XmlTextWriter classes. These classes provide a fast, non-cached, forward-only method for reading, creating and writing an XML document.

.NET datasets are objects used to store data retrieved from databases, XML documents, and other sources. A dataset's ReadXml method retrieves the data within an XML file and stores it in the dataset. Datasets can be bound to interface controls that display and allow editing the data. Any data stored in a dataset, including data retrieved from databases and other sources, can be written out in XML format by calling the dataset's WriteXml method.

The XmlReader class and XmlReaderSettings classes together allow validating XML documents from XSDs.

,NET supplies an XSLT processor in its XslCompiledTransform class making XSLTs an option for transforming an XML document into the different formats that are needed by applications using different data storage schemas and technologies.

Other technologies exist to do similar XML tasks. The intent of this paper isn't to place Microsoft's .NET technologies above others. It is only to make the reader aware that there is strong XML support in .NET and this is a viable choice to consider when introducing students to XML with hands on exercises and assignments.

5 REFERENCES

Dietrich, S. W., Urban, S. D., Ma, H., Xiao, Y., Patel, S. Exploring XML for Data Exchange in the Context of an Undergraduate Database Curriculum. In *Proceedings of the ACM Technical Symposium on Computer Science Education* (*SIGCSE '05*) (St. Louis, Feb. 2005). ACM Press, New York, NY, 2005, 53-57.

Wagner, P. and Moore, T. Integrating XML into a Database Systems Course. In Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE '03), (Nevada, Feb. 2003). ACM Press, New York, NY, 2001, 26-60.