**The laboratory work 13**

**Adding information to XML files**

**Introduction**

In this article, you will see how to read and write XML documents in Microsoft .NET using C# language.

First, I will discuss XML .NET Framework Library namespace and classes. Then, you will see how to read and write XML documents. At the end of this article, I will show you how to use ADO.NET and XML .NET models to read and write XML documents from relational databases and vice versa.

**Microsoft .NET XML Namespaces and Classes**

Before starting to work with XML documents in .NET Framework, it is important to know about .NET namespace and classes provided by .NET Runtime Library. .NET provides five namespace - System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, and System.Xml.Xsl to support XML classes.

The System.Xml namespace contains major XML classes. This namespace contains many classes to read and write XML documents. In this article, we are going to concentrate on the reader and writing class. These reader and writer classes are used to read and write XML documents. These classes are -

* XmlReader
* XmlTextReader
* XmlValidatingReader
* XmlNodeReader
* XmlWriter
* XmlTextWriter

 As you can see, there are four reader and two writer classes.

The XmlReader class is an abstract bases class and contains methods and properties to read a document. The Read method reads a node in the stream. Besides reading functionality, this class also contains methods to navigate through document nodes. Some of these methods are MoveToAttribute, MoveToFirstAttribute, MoveToContent, MoveToFirstContent, MoveToElement and MoveToNextAttribute. ReadString, ReadInnerXml, ReadOuterXml, and ReadStartElement are more read methods. This class also has a method called Skip to skip the current node and move to the next one. We'll see these methods in our sample example.

The XmlTextReader, XmlNodeReader, and XmlValidatingReader classes are derived from XmlReader class. As their name explains, they are used to read text, nodes, and schemas.

The XmlWrite class contains functionality to write data to XML documents. This class provides many writing methods to write XML document items. This class is the base class for the XmlTextWriter class, which we'll be using in our sample example.

The **XmlNode** class plays an important role. However, this class represents a single node of XML that could be the root node of an XML document and could represent the entire file. This class is an abstract base class for many useful classes for inserting, removing, and replacing nodes, navigating through the document. It also contains properties to get a parent or child, name, last child, node type, and more. Three major classes derived from XmlNode are XmlDocument, XmlDataDocument and XmlDocumentFragment. XmlDocument class represents an XML document and provides methods and properties to load and save a document. It also provides functionality to add XML items such as attributes, comments, spaces, elements, and new nodes. The Load and LoadXml methods can be used to load XML documents and Save method to save a document respectively. XmlDocumentFragment class represents a document fragment, which can be used to add to a document. The XmlDataDocument class provides methods and properties to work with ADO.NET data set objects.

In spite of the above-discussed classes, the System.Xml namespace contains more classes. A few of them are XmlConvert, XmlLinkedNode, and XmlNodeList.

The next namespace in the Xml series is System.Xml.Schema. It classes to work with XML schemas such as XmlSchema, XmlSchemaAll, XmlSchemaXPath, and XmlSchemaType.

The System.Xml.The serialization namespace contains classes that are used to serialize objects into XML format documents or streams.

The System.Xml.XPath Namespace contains XPath-related classes to use XPath specifications. This namespace has the following classes -XPathDocument, XPathExression, XPathNavigator, and XPathNodeIterator. With the help of XpathDocument, XpathNavigator provides a fast navigation through XML documents. This class contains many Move methods to move through a document.

The System.Xml.Xsl namespace contains classes to work with XSL/T transformations.

**How to read XML Documents?**

In my sample application, I'm using books.xml to read and display its data through XmlTextReader. This file comes with VS.NET samples. You can search this on your machine and change the path of the file in the following line:

XmlTextReader textReader = new XmlTextReader("C:\\books.xml");

C#

Or you can use any XML file.

The XmlTextReader, XmlNodeReader, and XmlValidatingReader classes are derived from XmlReader class. Besides XmlReader methods and properties, these classes also contain members to read text, node, and schemas respectively. I am using the XmlTextReader class to read an XML file. You read a file by passing the file name as a parameter in the constructor.

XmlTextReader textReader = new XmlTextReader("C:\\books.xml");

C#

After creating an instance of XmlTextReader, you call the Read method to start reading the document. After the read method is called, you can read all information and data stored in a document. XmlReader class has properties such as Name, BaseURI, Depth, LineNumber, and so on.

List 1 reads a document and displays node information using these properties.

**Example 1**

In this sample example, I read an XML file using XmlTextReader and called the Read method to read its nodes one by one until the end of the file and display the contents to the console output.

using System;

using System.Xml;

namespace ReadXml1 {

class Class1 {

static void Main(string[] args) {

// Create an isntance of XmlTextReader and call Read method to read the file

XmlTextReader textReader = new XmlTextReader("C:\\books.xml");

textReader.Read();

// If the node has value

while (textReader.Read()) {

// Move to fist element

textReader.MoveToElement();

Console.WriteLine("XmlTextReader Properties Test");

Console.WriteLine("===================");

// Read this element's properties and display them on console

Console.WriteLine("Name:" + textReader.Name);

Console.WriteLine("Base URI:" + textReader.BaseURI);

Console.WriteLine("Local Name:" + textReader.LocalName);

Console.WriteLine("Attribute Count:" + textReader.AttributeCount.ToString());

Console.WriteLine("Depth:" + textReader.Depth.ToString());

Console.WriteLine("Line Number:" + textReader.LineNumber.ToString());

Console.WriteLine("Node Type:" + textReader.NodeType.ToString());

Console.WriteLine("Attribute Count:" + textReader.Value.ToString());

}

}

}

}

C#

The NodeType property of XmlTextReader is important when you want to know the content type of a document. The XmlNodeType enumeration has a member for each type of XML item, such as Attribute, CDATA, Element, Comment, Document, DocumentType, Entity, ProcessInstruction, WhiteSpace, and so on.

List 2 code sample reads an XML document, finds a node type, and writes information at the end with how many node types a document has.

**Example 2**

In this sample example, I read an XML file using XmlTextReader and called the Read method to read its nodes one by one until the end of the file. After reading a node, I check its NodeType property to find the node write node contents to the console, and keep track of a number of particular types of nodes. In the end, I display the total number of different types of nodes in the document.

using System;

using System.Xml;

namespace ReadingXML2 {

class Class1 {

static void Main(string[] args) {

int ws = 0;

int pi = 0;

int dc = 0;

int cc = 0;

int ac = 0;

int et = 0;

int el = 0;

int xd = 0;

// Read a document

XmlTextReader textReader = new XmlTextReader("C:\\books.xml");

// Read until end of file

while (textReader.Read()) {

XmlNodeType nType = textReader.NodeType;

// If node type us a declaration

if (nType == XmlNodeType.XmlDeclaration) {

Console.WriteLine("Declaration:" + textReader.Name.ToString());

xd = xd + 1;

}

// if node type is a comment

if (nType == XmlNodeType.Comment) {

Console.WriteLine("Comment:" + textReader.Name.ToString());

cc = cc + 1;

}

// if node type us an attribute

if (nType == XmlNodeType.Attribute) {

Console.WriteLine("Attribute:" + textReader.Name.ToString());

ac = ac + 1;

}

// if node type is an element

if (nType == XmlNodeType.Element) {

Console.WriteLine("Element:" + textReader.Name.ToString());

el = el + 1;

}

// if node type is an entity\

if (nType == XmlNodeType.Entity) {

Console.WriteLine("Entity:" + textReader.Name.ToString());

et = et + 1;

}

// if node type is a Process Instruction

if (nType == XmlNodeType.ProcessingInstruction)

{

myXMLRichTextBox.AppendText("Process Instruction:" + textReader.Name.ToString() + Environment.NewLine); pi = pi + 1;

}

// if node type a document

if (nType == XmlNodeType.DocumentType) {

Console.WriteLine("Document:" + textReader.Name.ToString());

dc = dc + 1;

}

// if node type is white space

if (nType == XmlNodeType.Whitespace) {

Console.WriteLine("WhiteSpace:" + textReader.Name.ToString());

ws = ws + 1;

}

}

// Write the summary

Console.WriteLine("Total Comments:" + cc.ToString());

Console.WriteLine("Total Attributes:" + ac.ToString());

Console.WriteLine("Total Elements:" + el.ToString());

Console.WriteLine("Total Entity:" + et.ToString());

Console.WriteLine("Total Process Instructions:" + pi.ToString());

Console.WriteLine("Total Declaration:" + xd.ToString());

Console.WriteLine("Total DocumentType:" + dc.ToString());

Console.WriteLine("Total WhiteSpaces:" + ws.ToString());

}

}

}

C#

**How to Write XML Documents?**

XmlWriter class contains the functionality to write to XML documents. It is an abstract base class used through XmlTextWriter and XmlNodeWriter classes. It contains methods and properties to write to XML documents. This class has several Writexxx methods to write every type of item of an XML document. For example, WriteNode, WriteString, WriteAttributes, WriteStartElement, and WriteEndElement are some of them. Some of these methods are used in a start and end pair. For example, to write an element, you need to call WriteStartElement and then write a string followed by WriteEndElement.

Besides many methods, this class has three properties. WriteState, XmlLang, and XmlSpace. The WriteState gets and sets the state of the **XmlWriter class**.

Although it's not possible to describe all the Writexxx methods here, let's see some of them.

The first thing we need to do is create an instance of XmlTextWriter using its constructor. XmlTextWriter has three overloaded constructors, which can take a string, stream, or a TextWriter as an argument. We'll pass a string (file name) as an argument, which we're going to create in C:\ root.

In my sample example, I create a file myXmlFile.xml in C:\\ root directory.

// Create a new file in C:\\ dir

XmlTextWriter textWriter = new XmlTextWriter("C:\\myXmFile.xml", null) ;

C#

After creating an instance, first thing you call us WriterStartDocument. When you're done writing, you call the WriteEndDocument and TextWriter's Close method.

textWriter.WriteStartDocument();

textWriter.WriteEndDocument();

textWriter.Close();

C#

The WriteStartDocument and WriteEndDocument methods open and close a document for writing. You must have to open a document before starting to write to it. The writeComment method writes a comment to a document. It takes only one string type of argument. The writeString method writes a string to a document. With the help of WriteString, WriteStartElement, and WriteEndElement methods pair can be used to write an element to a document. The WriteStartAttribute and WriteEndAttribute pair writes an attribute.

WriteNode is more write method that writes an XmlReader to a document as a node of the document. For example, you can use WriteProcessingInstruction and WriteDocType methods to write ProcessingInstruction and DocType items of a document.

//Write the ProcessingInstruction node

string PI= "type='text/xsl' href='book.xsl'"

textWriter.WriteProcessingInstruction("xml-stylesheet", PI);

//'Write the DocumentType node

textWriter.WriteDocType("book", Nothing, Nothing, "<!ENTITY h 'softcover'>");

C#

The below sample example summarizes all these methods and creates a new xml document with some items in it, such as elements, attributes, strings, comments, and so on. See Listing 5-14. In this sample example, we create a new xml file c:\xmlWriterText.xml. In this sample example, We create a new xml file c:\xmlWriterTest.xml using XmlTextWriter:

After that, we add comments and elements to the document using Write methods. After that, we read our books.xml xml file using XmlTextReader and add its elements to xmlWriterTest.xml using XmlTextWriter.

**Example 3**

In this sample example, I create a new file, myxmlFile.xml, using XmlTextWriter and use its various write methods to write XML items.

using System;

using System.Xml;

namespace ReadingXML2 {

class Class1 {

static void Main(string[] args) {

// Create a new file in C:\\ dir

XmlTextWriter textWriter = new XmlTextWriter("C:\\myXmFile.xml", null);

// Opens the document

textWriter.WriteStartDocument();

// Write comments

textWriter.WriteComment("First Comment XmlTextWriter Sample Example");

textWriter.WriteComment("myXmlFile.xml in root dir");

// Write first element

textWriter.WriteStartElement("Student");

textWriter.WriteStartElement("r", "RECORD", "urn:record");

// Write next element

textWriter.WriteStartElement("Name", "");

textWriter.WriteString("Student");

textWriter.WriteEndElement();

// Write one more element

textWriter.WriteStartElement("Address", "");

textWriter.WriteString("Colony");

textWriter.WriteEndElement();

// WriteChars

char[] ch = new char[3];

ch[0] = 'a';

ch[1] = 'r';

ch[2] = 'c';

textWriter.WriteStartElement("Char");

textWriter.WriteChars(ch, 0, ch.Length);

textWriter.WriteEndElement();

// Ends the document.

textWriter.WriteEndDocument();

// close writer

textWriter.Close();

}

}

}

C#

**How to use XmlDocument?**

The XmlDocument class represents an XML document. This class provides similar methods and properties we've discussed earlier in this article.

Load and LoadXml are two useful methods of this class. A Load method loads XML data from a string, stream, TextReader, or XmlReader. LoadXml method loads XML documents from a specified string. Another useful method of this class is Save. Using the Save method, you can write XML data to a string, stream, TextWriter, or XMLWriter.

**Example 4**

This tiny sample example is pretty easy to understand. We call the LoadXml method of XmlDocument to load an XML fragment and call Save to save the fragment as an XML file.

//Create the XmlDocument.

XmlDocument doc = new XmlDocument();

doc.LoadXml(("<Student type='regular' Section='B'><Name>Tommy

ex</Name></Student>"));

//Save the document to a file.

doc.Save("C:\\std.xml");

C#

You can also use the Save method to display contents on the console if you pass the Console.Out as a parameter. For example:

doc.Save(Console.Out);

C#

**Example 5**

Here is one example of how to load an XML document using XmlTextReader. In this sample example, we read the books.xml file using XmlTextReader and call its Read method. After that, we call XmlDocumetn's Load method to load XmlTextReader contents to XmlDocument and call the Save method to save the document. Passing Console.Out as a Save method argument displays data on the console

XmlDocument doc = new XmlDocument();

//Load the the document with the last book node.

XmlTextReader reader = new XmlTextReader("c:\\books.xml");

reader.Read();

// load reader

doc.Load(reader);

// Display contents on the console

doc.Save(Console.Out);

C#

**Writing Data from a Database to an XML Document**

Using XML and ADO.NET mode, reading a database and writing to an XML document and vice versa is not a big deal. In this section of this article, you will see how to read a database table's data and write the contents to an XML document.

The DataSet class provides a method to read a relational database table and write this table to an XML file. You use the WriteXml method to write a dataset data to an XML file.

In this sample example, I have used the commonly used Northwind database that comes with Office 2000 and later versions. You can use any database you want. The only thing you need to do is just chapter the connection string and SELECT SQ L query.

**Example 6**

In this sample, I create a data adapter object and select all records of the Customers table. After that, I can use the method to fill a dataset from the data adapter.

In this sample example, I have used OldDb data provides. You need to add a reference to the System.Data.OldDb namespace to use OldDb data adapters in your program. As you can see from Sample Example 6, first, I create a connection with the Northwind database using OldDbConnection. After that, I create a data adapter object by passing a SELECT SQL query and connection. Once you have a data adapter, you can fill a dataset object using the Fill method of the data adapter. Then you can WriteXml method of DataSet, which creates an XML document and writes its contents to the XML document. In our sample, we read Customers' table records and write DataSet contents to OutputXml.Xml file in C:\ dir.

using System;

using System.Xml;

using System.Data;

using System.Data.OleDb;

namespace ReadingXML2 {

class Class1 {

static void Main(string[] args) {

// create a connection

OleDbConnection con = new OleDbConnection();

con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Northwind.mdb";

// create a data adapter

OleDbDataAdapter da = new OleDbDataAdapter("Select \* from Customers", con);

// create a new dataset

DataSet ds = new DataSet();

// fill dataset

da.Fill(ds, "Customers");

// write dataset contents to an xml file by calling WriteXml method

ds.WriteXml("C:\\OutputXML.xml");

}

}

}

C#

**Summary**

.NET Framework Library provides good support to work with XML documents. The XmlReader, XmlWriter, and their derived classes contain methods and properties to read and write XML documents. With the help of the XmlDocument and XmlDataDocument classes, you can read the entire document. The Load and Save method of XmlDocument loads a reader or a file and saves documents respectively. ADO.NET provides functionality to read a database and write its contents to the XML document using data providers and a DataSet object.