F

Understanding XML

Extensible Markup Language (XML) is a Data Description Language (DDL) developed by World Wide Web Consortium (W3C), which is used by ADO.NET (ADO stands for ActiveX Data Objects) as a format to store and transfer data. XML is a simple and flexible language, which is widely used to exchange a large variety of data, such as text, images, and movie clips over the Internet. The following is the list of some standards provided by W3C standards that are supported by .NET Framework 4.0:

- XML 1.0 (`www.w3.org/TR/1998/REC-xml-19980210`) with Document Type Definition (DTD) support
- XML Namespaces (`www.w3.org/TR/REC-xml-names`) for both stream-level and DOM
- XML Schemas (www.w3.org/2001/XMLSchema)
- XPath expressions (www.w3.org/ TR /xpath)
- XSLT transformations (`www.w3.org/ TR /xslt`)
- DOM Level 1 Core (www.w3.org/ TR /REC-DOM-Level-1/)
- DOM Level 2 Core (www.w3.org/ TR / REC-DOM-Level-2-Core/)

SOAP 1.1 (www.w3.org/ TR /SOAP)

The Document Object Model (DOM) is a platform-independent interface, which allows programs and scripts to access and update the content and structure of documents dynamically. In .NET applications, you can implement DOM with the help of the `XmlDocument` class and serialize and deserialize objects using the `System.Xml.Serialization` namespace. .NET Framework 4.0 also provides a namespace called System.Xml, which includes classes that are used to work with XML.

In this appendix, you first learn about how .NET Framework 4.0 supports XML. Next, you learn about the `System.Xml` namespace and how to work with streamed XML data. You also learn to implement DOM in .NET, XPath, and Extensible Stylesheet Language Transformations (XSLT) in .NET Framework 4.0. Finally, you learn to use XML with ADO.NET and serialize objects with XML.

Let's start our discussion with the System.Xml namespace.

Introducing the System.Xml Namespace

The `System.Xml` namespace in .NET Framework 4.0 contains classes that are used to process XML.

Table F.1 lists the noteworthy classes contained in the `System.Xml` namespace:

Table F.1: Noteworthy Classes Contained in the System.Xml Namespace

Class	Description
XmlReader	Provides forward-only and read-only access to XML data. It is an abstract reader class.
XmlWriter	Provides fast non-cached XML data in the form of stream or file format. It is an abstract

Table F.1: Noteworthy Classes Contained in the System.Xml Namespace	
Class	**Description**
	writer class.
XmlTextReader	Extends the XmlReader class and provides fast and forward-only stream access to XML data.
XmlTextWriter	Extends the XmlWriter class and provides fast and forward-only generation of XML streams.
XmlNode	Represents a single node in an XML document. This abstract class is a base class for many classes in the XML namespace.
XmlDocument	Extends the XmlNode class and implements DOM. It provides a tree structure for an XML document in memory and enables the user to navigate and edit XML documents.
XmlResolver	Resolves external XML-based resources, such as DTD and schema references. This abstract class is also used for processing the <xml:include> and <xml:import> elements.
XmlUrlResolver	Extends the XmlResolver class and resolves resources named by a Uniform Resource Identifier (URI).

Next, let's learn to work with streamed XML data in .NET applications.

Working with Streamed XML Data

Prior to the introduction of DOM, Simple Application Programming Interface for XML (SAX) was used to access information stored in XML documents. In the SAX model, only the `XmlReader` and `XmlWriter` classes were available to work with XML data. Memory requirements for the SAX model are not as high as the memory requirements for a streaming model. The SAX model, on the other hand, does not provide navigation flexibility and read or write capabilities that are available in DOM.

`XmlWriter`-based classes are used to generate XML documents. You can use classes derived from the `XmlReader` and `XmlWriter` abstract classes to work with XML in your applications. Classes derived from the `XmlReader` abstract class are as follows:

- `XmlTextReader`—Works either with a stream-based object or with `Textreader/TextWriter` objects from the `System.IO` namespace
- `XmlNodeReader`—Uses `XmlNode` class as its source instead of a stream
- `XmlValidatingReader`—Adds DTD and schema validation; thereby, offering data validation

The `XmlTextWriter` class is a class that is derived from the `XmlWriter` abstract class. Similar to the `XmlTextReader` class, this class also works either with a stream-based object or with the `TextReader/TextWriter` objects from the `System.IO` namespace.

Now, let's look at some of these classes in detail.

The XmlReader Class

The `XmlReader` class is used to read XML data in a fast, forward-only, and non-cached manner. The working of the `XmlReader` class is similar to the SAX model. However, there are few differences between the two. One of these differences is the type of model the `XmlReader` class and SAX model use to read data from an XML document. The SAX model uses a push model to read XML data, while the `XmlReader` class uses a pull model for this purpose.

NOTE

In the push model, the data is pushed by the SAX parser into the event handler; whereas, in the pull model, only the selected data is pulled by the parser for processing. The pull model is used by an XmlReader class, which allows you to sequentially access the information contained in an XML file.

One of the disadvantages of the push model is that, in this model, data is pushed out to the application and the developer of the application needs to be ready to receive it. Another disadvantage is that all the data pushed out to the application needs to be processed by the application, whether it is required or not. On the other hand, in the pull model, data is automatically pulled into the application that is requesting it and you do not need to process all the data sent to the application. You can select the data that you want and the application will pull only that data for processing.

To be able to work with the `XmlReader` class, you need to import the following namespace in your C# application:

```
using System.Xml;
```

To be able to work with the `XmlReader` class, you need to import the following namespace in your VB application:

```
Imports System.Xml
```

You can read data from an XML document in a number of ways. The methods that you can use for reading data from an XML document are given in Table F.2:

Table F.2: Noteworthy Methods from the XmlReader Class for Reading XML Data

Method	Description
Read	Reads the next node in an Xml document. This method returns true if the next node in an Xml document is successfully read. You can then use either the HasValue method to verify if the node has a value or the HasAttributes method to verify if the node has any attribute.
ReadStartElement	Verifies whether the current node is a first element or a start tag and then advances the reader to the next node. An exception of the XmlException type is raised if the current node is not the start element. The effect of calling this method is the same as of calling the IsStartElement method, followed by the Read method.
ReadString	Reads in the text data from an element. It returns a string object containing the data.
ReadElementString	Works in the same manner as the ReadString method does. The only difference is that you can optionally pass the name of the element to this method.

Next, let's learn how to perform validation on an XML document using the XmlReader class.

Validation with the XmlReader Class

To validate an XML document while reading, with the help of the `XmlReader` class, you need to create an object of the `XmlReaderSettings` class. Thereafter, you need to add the XSD schema to the `XmlSchemaSet class`, by setting the `Schemas` property of the `XmlReaderSettings` class. You also need to set the `XsdValidate` property of the `XmlReaderSettings` class to `True`, which has the value `False`, by default.

Now, let's create an XML file called Authors.xml to read all the element data from an XML file into our Web application. You can also read data for a single element by specifying the name of the element, or you can read all the attribute values from an XML file.

Listing F.1 shows the code for the `Authors.xml` file for reading XML data:

Listing F.1: Showing the Code for the `Authors.xml` File

```
<?xml version="1.0"?>
<authorslist>
  <authors au_id="409-56-7008">
        <au_lname>Tom</au_lname>
        <au_fname>Abraham</au_fname>
  </authors>
  <authors au_id="648-92-1872">
        <au_lname>Blotchet-Halls</au_lname>
        <au_fname>Reginald</au_fname>
  </authors>
  <authors au_id="238-95-7766">
        <au_lname>Carson</au_lname>
        <au_fname>Cheryl</au_fname>
```

```
      </authors>
      <authors au_id="722-51-5454">
            <au_lname>DeFrance</au_lname>
            <au_fname>Michel</au_fname>
      </authors>
      <authors au_id="427-17-2319">
            <au_lname>Dull</au_lname>
            <au_fname>Ann</au_fname>
      </authors>
      <authors au_id="213-46-8915">
            <au_lname>Green</au_lname>
            <au_fname>Marjorie</au_fname>
      </authors>
      <authors au_id="527-72-3246">
            <au_lname>Greene</au_lname>
            <au_fname>Morningstar</au_fname>
      </authors>
      <authors au_id="472-27-2349">
            <au_lname>Gringlesby</au_lname>
            <au_fname>Burt</au_fname>
      </authors>
      <authors au_id="846-92-7186">
            <au_lname>Hunter</au_lname>
            <au_fname>Sheryl</au_fname>
      </authors>
      <authors au_id="756-30-7391">
            <au_lname>Karsen</au_lname>
            <au_fname>Livia</au_fname>
      </authors>
    </authorslist>
```

To display the contents of an XML file in your application, create an empty ASP.NET Web application named `WorkingWithStreamedXML`. This application can be found in the CD-ROM as `WorkingWithStreamedXMLCS` for C# and `WorkingWithStreamedXMLVB` for VB. Now add a Web form and the Authors.xml file to your `WorkingWithStreamedXML` Web application. Then, add a `ListBox` control and three Button controls to the Default.aspx page of the `WorkingWithStreamedXML` Web application.

Listing F.2 shows the code for the `Default.aspx` page for reading XML data:

Listing F.2: Showing the Code for the `Default.aspx` Page

In VB

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="MyWebForm.aspx.vb"
  Inherits="WorkingWithStreamedXMLVB.MyWebForm" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <h1>
                          ASP.NET 4.0 Black Book
                </h1>
            <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
                Font-Underline="True" Text="Working with Streamed XML"></asp:Label>
                <br />
                <br />
                </div>
                <asp:ListBox ID="ListBox1" runat="server" Height="318px" Width="557px">
                </asp:ListBox>
                <br />
```

```
                    <br />
                    <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
                    Text="Show All Elements"
                    Font-Bold="True" />
                        
                    <asp:Button ID="Button2" runat="server" onclick="Button2_Click"
                    Text="Show Last Names"
                    Font-Bold="True" />
                        
                    <asp:Button ID="Button3" runat="server" onclick="Button3_Click"
                    Text="Show All Attributes"
                    Font-Bold="True" />
                    <br />
                    <br />
                    <br />
                    <br />
                    <div id="footer">
                            <p class="left">
                            All content copyright &copy; Kogent Solutions Inc.</p>
                   </div>
        </form>
    </body>
    </html>
```

In C#

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="MyWebForm.aspx.cs"
      Inherits="WorkingWithStreamedXMLCS.MyWebForm" %>

    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
        <title></title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
        <h1>
                            ASP.NET 4.0 Black Book
                    </h1>
                <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
                    Font-Underline="True" Text="Working with Streamed XML"></asp:Label>
                    <br />
                    <br />
                    </div>
                    <asp:ListBox ID="ListBox1" runat="server" Height="318px" Width="557px">
                    </asp:ListBox>
                    <br />
                    <br />
                    <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
                    Text="Show All Elements"
                    Font-Bold="True" />
                        
                    <asp:Button ID="Button2" runat="server" onclick="Button2_Click"
                    Text="Show Last Names"
                    Font-Bold="True" />
                        
                    <asp:Button ID="Button3" runat="server" onclick="Button3_Click"
                    Text="Show All Attributes"
                    Font-Bold="True" />
                    <br />
                    <br />
                    <br />
                    <br />
```

```
                    <div id="footer">
                            <p class="left">
                            All content copyright &copy; Kogent Solutions Inc.</p>
        </div>
        </form>
    </body>
    </html>
```

Now, add the code in the code-behind file of the `Default.aspx` page for reading XML data, as shown in Listing F.3:

Listing F.3: Showing the code for the Code-Behind File of the `Default.aspx Page`

In VB

```
    Imports System
    Imports System.Xml
    Public Class MyWebForm
        Inherits System.Web.UI.Page

        Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
            Me.Load
        End Sub
        Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
            Button1.Click
            ListBox1.Items.Clear()
            Dim xr As XmlReader = XmlReader.Create("D:\\Books\\Black Book\\ASP.NET
             4.0_BB\\Source Code\\Appendix
             F\\VB\\WorkingWithStreamedXMLVB\\WorkingWithStreamedXMLVB\\Authors.xml")
            Do While xr.Read()
                If xr.NodeType = XmlNodeType.Text Then
                    ListBox1.Items.Add(xr.Value)
                End If
            Loop
        End Sub

        Protected Sub Button2_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
            Button2.Click
            ListBox1.Items.Clear()
            Dim xr As XmlReader = XmlReader.Create("D:\\Books\\Black Book\ASP.NET
             4.0_BB\\Source Code\\Appendix
             F\\VB\\WorkingWithStreamedXMLVB\\WorkingWithStreamedXMLVB\\Authors.xml")
            Do While xr.Read()
                If xr.NodeType = XmlNodeType.Element AndAlso xr.Name = "au_lname" Then
                    ListBox1.Items.Add(xr.ReadElementString())
                Else
                    xr.Read()
                End If
            Loop
        End Sub

        Protected Sub Button3_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
            Button3.Click
            ListBox1.Items.Clear()
            Dim xr As XmlReader = XmlReader.Create("D:\\Books\\Black Book\\ASP.NET
             4.0_BB\\Source Code\\Appendix
             F\\VB\\WorkingWithStreamedXMLVB\\WorkingWithStreamedXMLVB\\Authors.xml")
            Do While xr.Read()
                If xr.NodeType = XmlNodeType.Element Then
                    For i As Integer = 0 To xr.AttributeCount - 1
                        ListBox1.Items.Add(xr.GetAttribute(i))
                    Next i
                End If
            Loop
        End Sub
    End Class
```

In C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Configuration;
using System.Data;
using System.Web.Security;
using System.Xml;
namespace WorkingWithStreamedXMLCS {
    public partial class MyWebForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
        protected void Button1_Click(object sender, EventArgs e)
  {
         ListBox1.Items.Clear();
         XmlReader xr = XmlReader.Create("D:\\Books\\Black Book\\ASP.NET 4.0_BB\\Source
         Code\\Appendix
         F\\CS\\WorkingWithStreamedXMLCS\\WorkingWithStreamedXMLCS\\Authors.xml");
         while (xr.Read())
         {
                 if (xr.NodeType == XmlNodeType.Text)
                 ListBox1.Items.Add(xr.Value);
         }
  }
  protected void Button2_Click(object sender, EventArgs e)
  {
         ListBox1.Items.Clear();
         XmlReader xr = XmlReader.Create("D:\\Books\\Black Book\\ASP.NET 4.0_BB\\Source
         Code\\Appendix
         F\\CS\\WorkingWithStreamedXMLCS\\WorkingWithStreamedXMLCS\\Authors.xml");
         while (xr.Read())
         {
                 if (xr.NodeType == XmlNodeType.Element && xr.Name == "au_lname")
                 ListBox1.Items.Add(xr.ReadElementString());
                 else
                 xr.Read();
         }
  }
  protected void Button3_Click(object sender, EventArgs e)
  {
         ListBox1.Items.Clear();
         XmlReader xr = XmlReader.Create("D:\\Books\\Black Book\\ASP.NET 4.0_BB\\Source
         Code\\Appendix
         F\\CS\\WorkingWithStreamedXMLCS\\WorkingWithStreamedXMLCS\\Authors.xml");
         while (xr.Read())
         {
                 if (xr.NodeType == XmlNodeType.Element)
                 {
                         for (int i = 0; i < xr.AttributeCount; i++)
                         ListBox1.Items.Add(xr.GetAttribute(i));
                 }
         }
  }
    }
}
```

NOTE

Prior to the execution of the application, change the location of the XML file according to the location of the XML file on your system.

Now, run the `WorkingWithStreamedXML` Web application by pressing the F5 key and click the Show All Elements button. All elements are added to the `ListBox` control, as shown in Figure F.1:

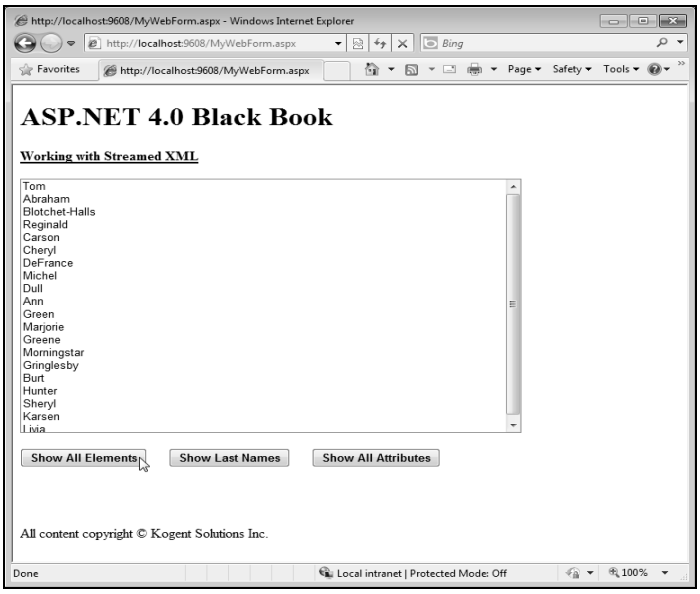

Figure F.1: Displaying All Elements of an XML Document in List Box

Next, click the Show Last Names button. All elements with the tag name `au_lname` are added to the `ListBox` control, as shown in Figure F.2:

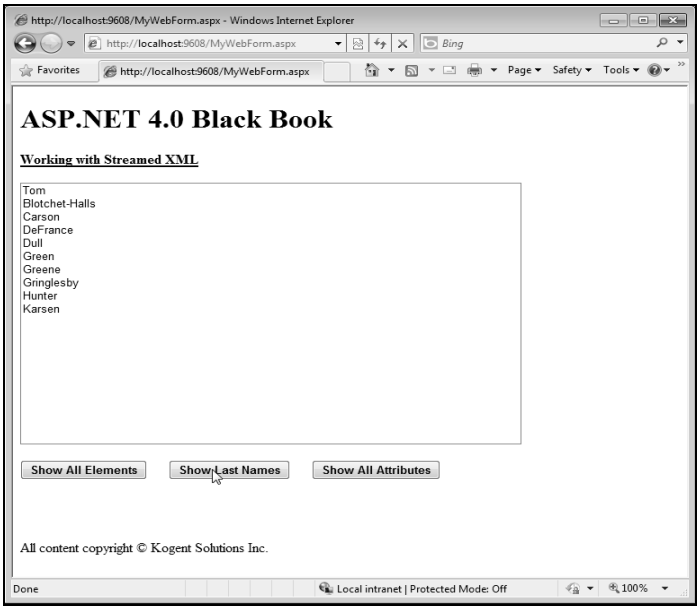

Figure F.2: Displaying All au_lname Elements of an XML Document in List Box

Now, click the Show All Attributes button. All the attributes of the XML document are added to the `ListBox` control, as shown in Figure F.3:

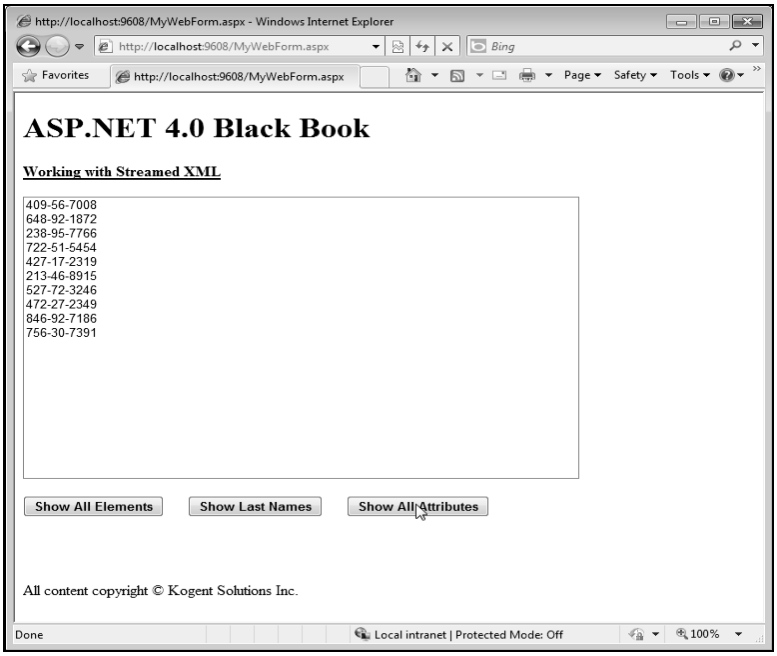

Figure F.3: Displaying All Attributes of an XML Document in List Box

Next, let's learn about the XmlWriter class in .NET Framework 4.0.

The XmlWriter Class

The `XmlWriter` class is used to write XML to a stream, a file, or a `TextWriter` object. This class works in a forward-only, non-cached manner. You can configure the `XmlWriter` object up to a large extent.

Table F.3 shows the noteworthy methods of the XmlWriter class that are used to create an XML document:

Table F.3: Noteworthy Methods of the XmlWriter Class

Method	Description
`Create`	Creates an instance of the XmlWriter class
`WriteAttributes`	Writes out all the attributes that are found at the current position in the XmlReader object to the XmlWriter object
`WriteAttributeString`	Writes out an attribute with a specified local name and value
`WriteBase64`	Encodes the binary bytes that you specify as Base64 and writes out the resulting text
`WriteCData`	Writes out the character data (CDATA) block containing the text that you specify
`WriteCharEntity`	Generates a character entity for a Unicode character value that you specify
`WriteChars`	Writes text one buffer at a time
`WriteComment`	Writes an XML comment containing the text that you specify
`WriteDocType`	Writes the DOCTYPE declaration with the specified name and optional attributes
`WriteElementString`	Writes an element with a local name and value that you specify
`WriteEndAttribute`	Closes the previous WriteStartAttribute call made by an object of the XmlWriter class
`WriteEndDocument`	Closes any open elements or attributes and puts the writer back in the Start state

Table F.3: Noteworthy Methods of the XmlWriter Class	
Method	**Description**
`WriteEndElement`	Closes an element opened using the WriteStartElement method of the XmlWriter class
`WriteNode`	Copies everything from the reader to the current instance of the XmlWriter class
`WriteStartAttribute`	Writes the start of an attribute with a local name that you specify
`WriteStartDocument`	Writes the XML declaration with the version "1.0"
`WriteStartElement`	Writes out a start tag with a local name that you specify

Now, let's create an empty ASP.NET Web application called XmlWriterApp to see how the XmlWriter class can be used to create an XML document. This application can be found on the CD-ROM as XmlWriterAppCS for C# and XmlWriterAppVB for VB. Add a Web form to your XmlWriterApp Web application. Add the code given in Listing F.4 to the code-behind file of the Default.aspx page of the XmlWriterApp Web application to create an XML document using the XmlWriter class:

Listing F.4: Creating an XML File Using the XmlWriter Class

In VB

```
Imports System.Text
Imports System.Collections.Generic
Imports System.Linq
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Xml

Public Class MyWebForm
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
         Me.Load
        Dim docSettings As New XmlWriterSettings()
        docSettings.Indent = True
        docSettings.Encoding = Encoding.UTF8

        Dim docPath As String = Server.MapPath("Employees.xml")
        Try
            Using xr As XmlWriter = XmlWriter.Create(docPath, docSettings)
                xr.WriteStartDocument()
                xr.WriteComment("This File Displays Employee Details")
                xr.WriteStartElement("Employees")
                xr.WriteStartElement("Employee")
                xr.WriteAttributeString("ID", "Emp1")
                xr.WriteStartElement("Name")
                xr.WriteElementString("FirstName", "Gerry")
                xr.WriteElementString("LastName", "Simpson")
                xr.WriteEndElement()
                xr.WriteElementString("Age", "25")
                xr.WriteElementString("Designation", "Manager")
                xr.WriteStartElement("Location")
                xr.WriteElementString("City", "New Delhi")
                xr.WriteElementString("Country", "India")
                xr.WriteEndElement()
                xr.WriteEndElement()
                xr.WriteEndElement()
                xr.WriteEndDocument()
            End Using
            Response.Redirect("Employees.xml")
        Catch ex As Exception
            Response.Write(ex.Message)
        End Try
```

```
        End Sub

    End Class
```

In C#

```
    using System;
    using System.Text;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;
    using System.Web.UI;
    using System.Web.UI.WebControls;
    using System.Xml;

    namespace XmlWriterAppCS
    {
        public partial class MyWebForm : System.Web.UI.Page
        {
            protected void Page_Load(object sender, EventArgs e)
            {
                XmlWriterSettings docSettings = new XmlWriterSettings();
                docSettings.Indent = true;
                docSettings.Encoding = Encoding.UTF8;

                string docPath = Server.MapPath("Employees.xml");
                try
                {
                    using (XmlWriter xr = XmlWriter.Create(docPath, docSettings))
                    {
                        xr.WriteStartDocument();
                        xr.WriteComment("This File Displays Employee Details");
                        xr.WriteStartElement("Employees");
                        xr.WriteStartElement("Employee");
                        xr.WriteAttributeString("ID", "Emp1");
                        xr.WriteStartElement("Name");
                        xr.WriteElementString("FirstName", "Gerry");
                        xr.WriteElementString("LastName", "Simpson");
                        xr.WriteEndElement();
                        xr.WriteElementString("Age", "25");
                        xr.WriteElementString("Designation", "Manager");
                        xr.WriteStartElement("Location");
                        xr.WriteElementString("City", "New Delhi");
                        xr.WriteElementString("Country", "India");
                        xr.WriteEndElement();
                        xr.WriteEndElement();
                        xr.WriteEndElement();
                        xr.WriteEndDocument();
                    }
                    Response.Redirect("Employees.xml");
                }
                catch (Exception ex)
                {
                    Response.Write(ex.Message);
                }
            }
        }
    }
```

In Listing F.4, the object of the XmlWriterSettings class, docSettings is used to specify a set of features to be supported by the object of the XmlWriter class, xr. The object of the XmlWriter class, xr is created by invoking the XmlWriter.Create() method. In addition, the Response.Redirect() method is used to load the newly created XML file, Employees.xml in the Web browser.

Press the F5 key to run your XmlWriterApp Web application. The output appears, as shown in Figure F.4:

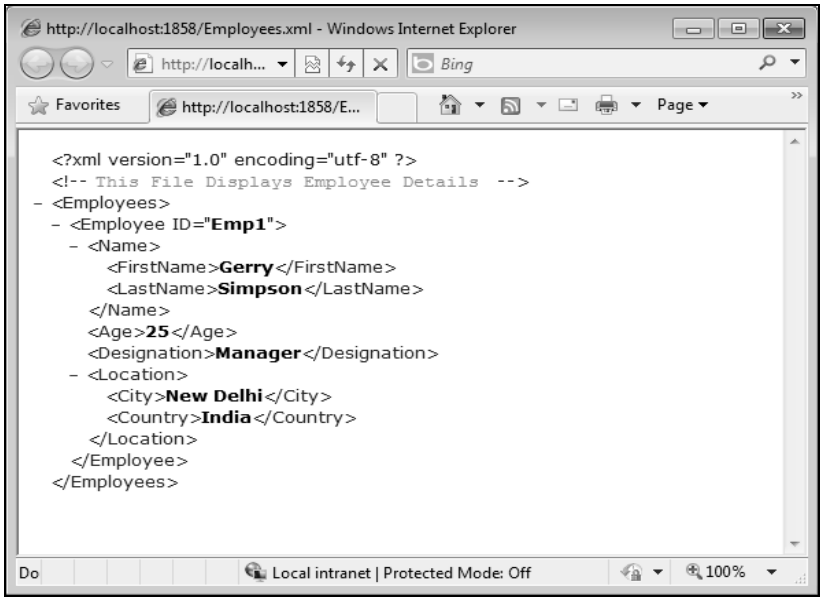

Figure F.4: Creating an XML File

Now, let's learn how to implement DOM in .NET Framework 4.0 using the classes contained in the `System.Xml` namespace.

Implementing DOM in .NET Framework 4.0

DOM is implemented in .NET Framework 4.0 by using the `XmlNode` abstract class and the `XmlNodeList` class. Implementation of DOM in .NET Framework 4.0 supports Core DOM Level 1 and Core DOM Level 2 standards developed by W3C. The `XmlNode` abstract class represents a node of an XML document. The `XmlNodeList` class is an ordered list of nodes, which supports indexed access. Any changes made to any node are immediately reflected in this list.

Table F.4 displays the classes derived from the `XmlNode` abstract class:

Table F.4: Classes Derived from the XmlNode Abstract Class

Class	Description
`XmlAttribute`	Represents an attribute object of an `XmlElement` object.
`XmlDocument`	Represents the entire XML document and implements the DOM Level 1 and DOM Level 2 specifications. Unlike the `XmlReader` and `XmlWriter` classes, this class gives you read and write capabilities and random access to the DOM tree.
`XmlDocumentFragment`	Represents a fragment of the document tree.
`XmlEntity`	Represents a parsed or unparsed entity node.
`XmlLinkedNode`	Contains the `NextSibling` and `PreviousSibling` properties, which are used for retrieving the node immediately after or before the current node.
`XmlNotation`	Contains a notation declared in a DTD or schema.

Table F.5 displays the classes derived from the `XmlLinkedNode` class:

Table F.5: Noteworthy Classes Derived from the XmlLinkedNode Class	
Class	**Description**
`XmlDeclaration`	Represents the declaration node (`<?xml version='1.0'…>`).
`XmlDocumentType`	Represents data relating to the DTD.
`XmlElement`	Represents an XML element object.
`XmlEntityReference`	Represents an entity reference node.
`XmlProcessingInstruction`	Contains an XML processing instruction.
`XmlCharacterData`	Provides text manipulation methods for other classes. It is an abstract class.

The `XmlDataDocument` class is derived from the `XmlDocument` class, which is listed in Table F.4. The `XmlDataDocument` class is built specifically for working with data associated with the `DataSet` object. This class has an overloaded constructor that takes a parameter, which is an object of type `DataSet`. When you create an object of the `XmlDataDocument` class by passing a parameter of type `DataSet` to its constructor, an XML document is created from the `DataSet` object. If you create an object of the `XmlDataDocument` class without passing any parameter to its constructor, the object will contain a `DataSet` with the name `NewDataSet` that has no `DataTables` in the `Tables` collection.

Table F.6 displays the classes derived from the `XmlCharacterData` abstract class:

Table F.6: Noteworthy Classes Derived from the XmlCharacterData Class	
Class	**Description**
`XmlCDataSection`	Represents a Character Data (`CData`) section of a document.
`XmlCommnet`	Represents an XML comment object.
`XmlSignificantWhitespace`	Represents a node with whitespace. With this class, the node is created only if the `PreserveWhiteSpace` flag is set to `True`.
`XmlWhitespace`	Represents a whitespace in element content. In this class also, the node is created only if the `PreserveWhiteSpace` flag is set to `True`.
`XmlText`	Represents the text content of an element or attribute.

Next, let's learn about the System.Xml.XPath and the System.Xml.Xsl namespaces.

Exploring XPath and XSLT in .NET Framework 4.0

The `System.Xml.XPath` namespace of .NET Framework 4.0 contains classes to select a subset of elements based on the element or attribute values. Another namespace of .NET Framework 4.0, named `System.Xml.Xsl`, contains classes used to transform a base document into another document of different structure or type.

Now, let's first learn about the System.Xml.XPath namespace in .NET Framework 4.0.

The System.Xml.XPath Namespace

The System.Xml.XPath namespace is used to retrieve a subset of elements based on a given criteria. In .NET Framework, you can import the `System.Xml.XPath` namespace to work with XPath, which facilitates fast processing. The `System.Xml.XPath` namespace provides a read-only view of the XML document with no editing rights. The classes contained in the `System.Xml.XPath` namespace are used to perform fast iteration and selections on an XML document.

Table F.7 displays the classes contained in the `System.Xml.XPath` namespace:

Table F.7: Noteworthy Classes Contained in the System.Xml.XPath Namespace	
Class	**Description**
XPathDocument	Provides a read-only view of the entire document
XPathNavigator	Provides navigation capabilities to an XmlDataDocument object
XPathNodeIterator	Provides iteration capabilities to a node set
XPathExpression	Represents a compiled XPath expression
XPathException	Represents an XPath exception class

Now, let's discuss some of the classes listed in Table F.7.

The XPathDocument Class

The XPathDocument class does not provide any functionality of the XmlDocument class. You have to use the XmlDocument class to obtain editing capabilities. To work with ADO.NET, you have to use the XmlDataDocument class. You can use the XPathDocument class to enable faster processing. The constructor of the XPathDocument class has many overloaded forms. You can pass the file name of an XML document, a TextReader object, an XmlReader object, or a Stream-based object to the constructor of the XPathDocument class.

The XPathNavigator Class

The XPathNavigator class contains methods that are used to move and select elements from an XML document. Table F.8 displays some methods from the XPathNavigator class, which are used to move the contents of an XML document:

Table F.8: Noteworthy Methods of the XPathNavigator Class	
Method	**Description**
MoveTo	Takes a parameter, which is an object of the XPathNavigator class. It is used to move the current cursor position to the position that is specified in an XPathNavigator object.
MoveToAttribute	Moves the current cursor position to the named attribute. It takes the attribute name and namespace as its parameters.
MoveToFirstAttribute	Moves the current cursor position to the first attribute in the current element. It returns True, if executed successfully.
MoveToNextAttribute	Moves the current cursor position to the next attribute in the current element. It returns True, if executed successfully.
MoveToFirst	Moves the current cursor position to the first sibling in the current node. It returns True, if executed successfully; otherwise, returns False.
MoveToNext	Moves the current cursor position to the next sibling in the current node. It returns True, if executed successfully.
MoveToPrevious	Moves the current cursor position to the previous sibling in the current node. It returns True, if executed successfully.
MoveToFirstChild	Moves the current cursor position to the child of the current element. It returns True, if executed successfully.
MoveToId	Moves the current cursor position to the element with the ID supplied as a parameter.
MoveToParent	Moves the current cursor position to the parent of the current node. It returns True, if executed successfully.
MoveToRoot	Moves the current cursor position to the root node of the document.

In addition to the methods listed in Table F.8, which are used to move the current cursor position inside the XML document, the `XPathNavigator` class has few more methods that are used to select elements from an XML document. All these methods return an `XPathNodeIterator` object. These methods include `Select()`, `SelectAncestors()`, and `SelectChildren()`. The `Select()` method takes the XPath expression as a parameter, while the other two methods take the `XPathNodeType` object as a parameter.

The XPathNodeIterator Class

The `XPathNodeIterator` class allows iteration over a selected set of nodes. The following are the properties defined by this class:

- `Count`—Represents the total number of nodes in the `XPathNodeIterator` object
- `Current`—Represents an `XPathNavigator` object pointing to the current node
- `CurrentPosition`—Represents the current position in the form of an integer

  The following methods are defined by this class:

  - `Clone`—Creates a new copy of the `XPathNodeIterator` object
  - `MoveNext`—Moves the current cursor-position to the next node that matches the XPath expression, which has created the `XPathNodeIterator` object

Next, let's learn about the System.Xml.Xsl namespace in .NET Framework 4.0.

The System.Xml.Xsl Namespace

The classes contained in the `System.Xml.Xsl` namespace are used to perform Extensible Stylesheet Language (XSL) transformations in .NET Framework 4.0. The contents of this namespace are available to the classes, which implement the `IXPathNavigable` interface. In .NET Framework 4.0, the classes including the `XmlDocument`, `XmlDataDocument`, and `XPathDocument` classes implement the `IXPathNavigable` interface. XSLT uses a streaming pull model in which data is pulled into an application; thereby, allowing you to perform several transformations together. A custom reader can be applied between transformations, if required.

Now, let's learn how to use XML with ADO.NET in our applications.

Using XML with ADO.NET

As already discussed, XML is a medium of transferring data to and from the data stores and applications. ADO.NET is primarily designed to work within the XML environment. ADO.NET provides some powerful features that are used to read and write XML documents. Some classes contained in the `System.Xml` namespace, such as `XmlDataDocument`, are also used to work with ADO.NET relational data.

You can create an XML file with data from an ADO.NET data source by first filling a `DataSet` object with the data from an ADO.NET data source and then calling the `WriteXml` method of the `DataSet` object. The `WriteXml` method has two overloaded forms, one of which takes a string with the file path and name, and the other takes one more parameter to specify the mode, which takes value from an `XmlWriteMode` enumeration. The `XmlWriteMode` enumeration can have any of the following three values:

- `IgnoreSchema`—Refers to the enumeration value that is used when you do not want to write an inline schema at the start of our XML file.
- `WriteSchema`—Refers to the enumeration value that is used when you want that an inline schema should be written at the start of our XML file.
- `DiffGram`—Refers to an XML document that contains data related to pre- and post-edit sessions. A `DiffGram` may include any type of changes made in data, such as addition and deletion of some data elements. The `DiffGram` is useful in audit trail for a commit/rollback process. You can implement commit or rollback features with the help of a `DiffGram`. A `DiffGram` comes as a built-in feature in most of the Database Management Systems (DBMSs) these days.

Now, that you know how to use XML with ADO.NET, let's proceed to learn how objects are serialized with XML.

Describing Object Serialization with XML

Retaining an object's data into the memory disk is called object serialization. In .NET Framework, an object is serialized to facilitate the conversion of XML documents into Common Language Runtime (CLR) objects and vice versa. This helps in converting XML documents into a form, which enables them to be easily processed using programming languages, such as VB.NET and C#. In object serialization, public properties and public fields of an object are converted into XML elements or attributes or both.

In .NET Framework 4.0, you have the `System.Xml.Serialization` namespace for object serialization. You can use classes contained in this namespace for serializing objects. The `System.Xml.Serialization` namespace contains a class, named `XmlSerializer`, which is widely used for serializing objects. If you wish to use the `XmlSerializer` class for serializing objects, then you need to first create an object of the `XmlSerializer` class by specifying the type of object that you want to serialize. After that, create a stream/writer object to write the file to a stream/document. Finally, call the `Serialize` method of the `XmlSerializer` class by passing two parameters. One of these parameters is a stream/writer object and the other is the object that you want to serialize.

To deserialize an object, create two objects, an `XmlSerializer` object and a stream/reader object and then call the `Deserialize` method of the `XmlSerializer` class by passing the stream/reader object as a parameter. The `Deserialize` method returns the deserialized object that needs to be cast to the correct type.

The object that you want to serialize and the XML document are linked with the help of custom attributes of C# or VB. An attribute is a type of declarative information that can be retrieved by CLR at runtime. These attributes describe how the object should be serialized. These attributes can be created using a tool, called `xsd.exe`, which is included in .NET Framework 3.5. The characteristics of `xsd.exe` are as follows:

- Generates an XML schema from an XML file
- Generates an XML schema from an External Data Representation (XDR) schema file
- Generates `DataSet` objects from an XSD schema file
- Generates an XSD file from classes you have already developed
- Generates schemas from types in compiled assemblies
- Generates runtime classes that have custom attributes for `XmlSerialization`
- Determines the programming language in which the generated code should be written

Now, let's create an empty ASP.NET Web application, named ObjectSerializationXML. This application can be found in CD-ROM as `ObjectSerializationXMLCS` for C# and `ObjectSerializationXMLVB` for VB. Now, add a `ListBox` control and two Button controls to a Web page either by dragging and dropping it from the Standard tab of the Toolbox or by double-clicking this control in the Toolbox.

Listing F.5 shows the code for the `Default.aspx` page for object serialization with XML:

Listing F.5: Showing the Code for the `Default.aspx` Page

In VB

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="MyWebForm.aspx.vb"
Inherits="ObjectSerializationXMLVB.MyWebForm" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <h1>
                        ASP.NET 4.0 Black Book
                </h1>
```

```
            <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
        Font-
        Underline="True" Text="Object Serialization with XML"></asp:Label>
        <br />
        <br />
        <asp:ListBox ID="ListBox1" runat="server" Height="105px" Width="302px">
        </asp:ListBox>
        <br />
        <br />
        <asp:Button ID="Button2" runat="server" Text="Serialize Data" Font-Bold="True"
        Height="26px"
        Width="136px" onclick="Button2_Click1" />
         
        <asp:Button ID="Button1" runat="server" Text="View XML File" Font-Bold="True"
        Height="26px" Width="136px" onclick="Button1_Click1" />
        <br />
        <br />
        <br />
        <br />
       <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
    </div>
    </form>
</body>
</html>
```

In C#

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="MyWebForm.aspx.cs"
Inherits="ObjectSerializationXMLCS.MyWebForm" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <h1>
                        ASP.NET 4.0 Black Book
                </h1>
            <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Size="Medium"
        Font-
        Underline="True" Text="Object Serialization with XML"></asp:Label>
        <br />
        <br />
        <asp:ListBox ID="ListBox1" runat="server" Height="105px" Width="302px">
        </asp:ListBox>
        <br />
        <br />
        <asp:Button ID="Button2" runat="server" Text="Serialize Data" Font-Bold="True"
        Height="26px"
        Width="136px" onclick="Button2_Click1" />
         
        <asp:Button ID="Button1" runat="server" Text="View XML File" Font-Bold="True"
        Height="26px" Width="136px" onclick="Button1_Click1" />
        <br />
        <br />
        <br />
        <br />
       <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
    </div>
```

```
        </form>
    </body>
    </html>
```

You can serialize objects of a class to XML and create objects from an XML document. To see how the processes of serialization and deserialization are carried out when you have access to the source code of the class, whose object you want to serialize, you need to perform two things. First of all, you need to define a class and also add attributes to it to be used in the process of serialization. Next, you need to add the code in the code-behind file of the `Default.aspx` page for object serialization with XML, as shown in Listing F.6:

Listing F.6: Showing the code for the Code-Behind File of the `Default.aspx Page`

In VB

```
Imports System
Imports System.Xml
Imports System.Xml.Serialization
Imports System.IO

Public Class MyWebForm
    Inherits System.Web.UI.Page
    <XmlRoot()> _
    Public Class Employee
        Private empID As Integer
        Private empName As String
        Private empSalary As Decimal
        <XmlElement()> _
        Public Property ID() As Integer
            Get
                Return empID
            End Get
            Set(ByVal value As Integer)
                empID = value
            End Set
        End Property
        <XmlElement()> _
        Public Property Name() As String
            Get
                Return empName
            End Get
            Set(ByVal value As String)
                empName = value
            End Set
        End Property
        <XmlElement()> _
        Public Property Salary() As Decimal
            Get
                Return empSalary
            End Get
            Set(ByVal value As Decimal)
                empSalary = value
            End Set
        End Property
    End Class
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
  Me.Load

    End Sub

    Protected Sub Button1_Click1(ByVal sender As Object, ByVal e As EventArgs) Handles
         Button1.Click
        Dim XMLPath As String = "D:\employee.xml"
        Dim empdoc As New XmlDocument()
        Response.ContentType = "text/xml"
```

```
            Try
                empdoc.Load(XMLPath)
                Response.Write(empdoc.InnerXml)
                Response.End()
            Catch ex As XmlException
                Response.Write("XMLException:" & ex.Message)
            End Try
        End Sub

        Protected Sub Button2_Click1(ByVal sender As Object, ByVal e As EventArgs) Handles
             Button2.Click
            Dim emp As New Employee()
            emp.ID = 101
            emp.Name = "Steve"
            emp.Salary = 6000
            Dim tw As TextWriter = New StreamWriter("D:\employee.xml")
            Dim xs As New XmlSerializer(GetType(Employee))
            xs.Serialize(tw, emp)
            tw.Close()
            Dim fs As New FileStream("D:\employee.xml", FileMode.Open)
            Dim newXs As New XmlSerializer(GetType(Employee))
            Dim emp1 As Employee = CType(newXs.Deserialize(fs), Employee)
            If emp1 IsNot Nothing Then
                ListBox1.Items.Add(emp1.ID.ToString())
                ListBox1.Items.Add(emp1.Name.ToString())
                ListBox1.Items.Add(emp1.Salary.ToString())
            End If
            fs.Close()
        End Sub
    End Class
```

In C#

```
    using System;
    using System.Configuration;
    using System.Data;
    using System.Linq;
    using System.Web;
    using System.Web.Security;
    using System.Web.UI;
    using System.Web.UI.HtmlControls;
    using System.Web.UI.WebControls;
    using System.Web.UI.WebControls.WebParts;
    using System.Xml.Linq;
    using System.Xml;
    using System.Xml.Schema;
    using System.Xml.Serialization;
    using System.IO;

    namespace ObjectSerializationXMLCS
    {
        public partial class MyWebForm : System.Web.UI.Page
        {
            protected void Page_Load(object sender, EventArgs e)
            {

            }
            [XmlRoot("employee")]
            public class Employee
            {
                private int empID;
                private string empName;
                private Decimal empSalary;
```

```
            [XmlElement("id")]
            public int ID
            {
                get
                {
                    return empID;
                }
                set
                {
                    empID = value;
                }
            }
            [XmlElement("name")]
            public string Name
            {
                get
                {
                    return empName;
                }
                set
                {
                    empName = value;
                }
            }
            [XmlElement("salary")]
            public decimal Salary
            {
                get
                {
                    return empSalary;
                }
                set
                {
                    empSalary = value;
                }
            }
        }
        protected void Button1_Click1(object sender, EventArgs e)
        {
            String XMLPath = "D:\\employee.xml";
            XmlDocument empdoc = new XmlDocument();
            Response.ContentType = "text/xml";
            try
            {
                empdoc.Load(XMLPath);
                Response.Write(empdoc.InnerXml);
                Response.End();
            }
            catch (XmlException ex)
            {
                Response.Write("XMLException:" + ex.Message);
            }
        }

        protected void Button2_Click1(object sender, EventArgs e)
        {
            Employee emp = new Employee();
            emp.ID = 101;
            emp.Name = "Steve";
            emp.Salary = 6000;
            TextWriter tw = new StreamWriter("D:\\employee.xml");
            XmlSerializer xs = new XmlSerializer(typeof(Employee));
```

```
                xs.Serialize(tw, emp);
                tw.Close();
                FileStream fs = new FileStream("D:\\employee.xml", FileMode.Open);
                XmlSerializer newXs = new XmlSerializer(typeof(Employee));
                Employee emp1 = (Employee)newXs.Deserialize(fs);
                if (emp1 != null)
                {
                    ListBox1.Items.Add(emp1.ID.ToString());
                    ListBox1.Items.Add(emp1.Name.ToString());
                    ListBox1.Items.Add(emp1.Salary.ToString());
                }
                fs.Close();
            }
        }
    }
```

In Listing F.6, the attribute containing the `XmlRoot` invocation, which is placed before the `Employee` class definition, specifies that this class will be the root element in the XML file that is generated after serialization. The attribute containing the `XmlElement` invocation specifies that the member contained inside that attribute is an XML element for the XML file generated after serialization.

To serialize an object with XML, you first need to create an object of the `Employee` class and also set its properties. You then have to create an object of the `StreamWriter` class by specifying the name of the XML file to be generated after serialization and assigning this object to a reference variable of the `TextWriter` class. Next, you need to create an object of the `XmlSerializer` class by specifying the type of object you want to serialize (`Employee`) and then call the `Serialize` method of the `XmlSerializer` object by passing two parameters to it. The first parameter to this method is the `TextWriter` object and the second parameter is the `Employee` object. After that the `Close` method of the `TextWriter` object is called. Finally, the data of the XML file, `employee.xml`, is displayed in a Web browser.

To deserialize the data, create an object of the `FileStream` class to read the serialized object, which is the XML file `employee.xml`. Next, create an object of the `XmlSerializer` class by specifying the type information of `Employee`. You then have to call the `Deserialize` method of the `XmlSerializer` object by passing it the `FileStream` object and assigning the value returned by the `Deserialize` method to a reference variable of the `Employee` class with the help of explicit typecasting. The data of the `Employee` object is then added to a Listbox. Finally, call the `Close` method of the `FileStream` object.

When you run the application, a file with the name `employee.xml` is created in the specified location given in the code. You can serialize the data contained inside the `employee.xml` by clicking the `Serialize Data` button in the Web page, as shown in Figure F.5:

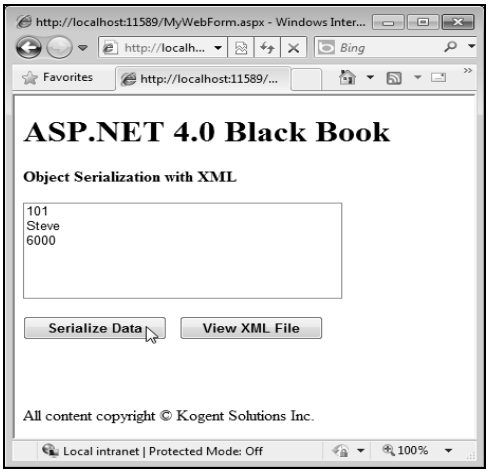

Figure F.5: Displaying Serialized Data of an XML File in a Listbox

If you want to see the XML file or deserialized data, click the `View XML File` button, as shown in Figure F.6:

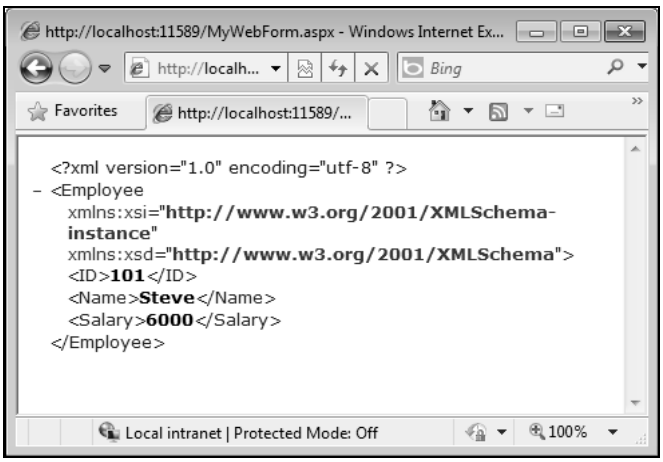

Figure F.6: Displaying XML View of Data in Browser

As shown in Figure F.6, the data is deserialized and displayed in the form of an XML file.