



# ASP.NET Developer's Cookbook

Copyright © 2003 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-32524-1

Library of Congress Catalog Card Number: 2003092438

Printed in the United States of America

First Printing: *June 2003*

Second Printing with corrections: *January 2004*

06 05 04 4 3 2

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales

1-317-581-3793

international@pearsontechgroup.com

## Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

## Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from programs accompanying it.

**Associate Publisher**  
Michael Stephens

**Acquisitions Editor**  
Neil Rowe

**Development Editor**  
Mark Renfrow

**Managing Editor**  
Charlotte Clapp

**Project Editor**  
Matthew Purcell

**Copy Editor**  
Kezia Endsley

**Indexer**  
John Sleeva

**Proofreader**  
Tracy Donhardt

**Technical Editor**  
Doug Holland

**Team Coordinator**  
Cindy Teeters

**Cover Designer**  
Alan Clements

**Page Layout**  
Julie Parks

**Graphics**  
Tammy Graham

# 12

## XML

### 12.0. Introduction

Extensible Markup Language, XML, has been a great success as a standard and is now prevalent throughout the information industry on all major platforms. Microsoft has thoroughly embraced XML, and provides a great deal of support for it in the .NET Framework. In fact, I'm composing this chapter within an XML document, which I am editing with Visual Studio .NET. You've already seen in Chapter 6, "ASP.NET Application Configuration," that ASP.NET configuration files are all XML formatted. In this chapter, you learn how to open, read, manipulate, transform, and save XML data. The `System.Xml` namespace holds the relevant classes that you will use to work with XML data, and the most commonly used ones are covered in this chapter.

### 12.1. Opening an XML File

You want to access an XML file on the server or at a particular URL.

#### Technique

You first create a generic reusable method that will take any sort XML string input, and return an appropriate `System.IO.Stream` with that XML properly loaded. The XML input can be in the following formats:

- Raw XML
- File system location
- HTTP URL

You must first determine whether the input received is raw XML. You can attempt to decide this by first determining whether the XML document provided starts

with `<?xml` or `<schema`. If this is true, you simply load the raw XML using the `System.IO.MemoryStream` class; if not true, you move to step two. The code is as follows:

```
Public Shared Function GetXmlDoc(ByVal xmlsource As String) _
As System.IO.Stream
    'first determine if xml source is actual raw xml
    'or if it is a url or file path
    Dim stream As System.IO.Stream = Nothing
    If xmlsource.StartsWith("<?xml") Or xmlsource.StartsWith("<schema") Then
        'raw xml
        stream = New System.IO.MemoryStream(
            System.Text.ASCIIEncoding.ASCII.GetBytes(xmlsource))
    End If
```

Now that you know the input is not raw XML, you still have one more decision to make. Is it a file system location or an actual HTTP URL? To determine this, you can take advantage of the `System.Uri` namespace provided by the framework. If you determine that it is a file system location, you can use the `System.IO.FileStream` class to load up the XML stream.

```
Dim xmluri As New System.Uri(xmlsource)
If xmluri.IsFile Then
    'file
    stream = New System.IO.FileStream(xmlsource, System.IO.FileMode.Open)
End If
```

You know that it is an HTTP URL (`xmluri.IsFile` returns `false`), so you need to first create an `HttpRequest` to download the stream off of the remote server.

```
Dim request As System.Net.HttpWebRequest =
CType(System.Net.WebRequest.Create(xmluri), System.Net.HttpWebRequest)
Dim response As System.Net.WebResponse = request.GetResponse()
stream = response.GetResponseStream()
```

The next step is to load the “stream” object into something you can work with for your XML needs. The object used here is `System.Xml.XmlDocument`.

```
Dim xmlDocStream As System.IO.Stream = GetXmlDoc(XmlSourceTextBox.Text)
Dim xmlSource As New System.Xml.XmlDocument()
xmlSource.Load(xmlDocStream)
ResultText.Text = xmlSource.InnerXml
```

## Comments

These four steps describe the process flow of loading an XML document when the source location and type are unknown. It’s a good idea to place the method in its own project (class library), so that you can reuse it in any of your projects in the future.

## 12.2. Finding a Particular Node in an XML Document

You want to locate a particular node in an XML document.

### Technique

This example builds on the previous one by enabling the ASP.NET application to select a node or a set of nodes from the XML document. Notice that this new ASP.NET Web Form has a few more TextBoxes and a new Query button. The first TextBox enables you to enter any XPath expression. The result appears in the second TextBox. You place the logic within the Query button.

The code is as follows:

```
Private Sub QueryButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs)
    Dim s as new System.Text.StringBuilder()

    If xmlSource Is Nothing Or xmlSource.InnerText = "" Then
        xmlSource.LoadXml (ResultText.Text)
    End If
    Try
        Dim nl As System.Xml.XmlNodeList = xmlSource.SelectNodes(XPathText.Text)
        Dim counter As Integer = 1
        Dim node As System.Xml.XmlNode
        For Each node In nl
            s.Append(Convert.ToString(counter) + "]" & node.InnerText & _
                System.Environment.NewLine)
            counter += 1
        Next node
        QueryResult.Text=s.ToString()
    Catch selectNodesError As Exception
        QueryResult.Text = selectNodesError.ToString()
    End Try
End Sub
```

### Comments

Test the form using these steps:

1. For the XML source, use <http://aspalliance.com/cookbook/samples/cdcatalog.xml>.
2. Press the Load Xml Document button.
3. Now use the following XPath expression: `/catalog/cd[title='Hide your heart']`.

4. Press the Query button.
5. You will notice that the expression resulted in the single node, which is placed in the last TextBox on the form.
6. Test different XPath expressions using this same procedure.

#### See Also

XPath Examples—[http://www.w3schools.com/xpath/xpath\\_examples.asp](http://www.w3schools.com/xpath/xpath_examples.asp)

## 12.3. Storing an XML File

You want to store an XML file on the file system.

### Technique

This example builds on Section 12.1 and enables the ASP.NET application to save the XML to the local file system. The code is as follows:

```

Private Sub SaveButton_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles SaveButton.Click
    If xmlSource Is Nothing Or xmlSource.InnerText = "" Then
        xmlSource.LoadXml(ResultText.Text)
    End If
    Try
        Dim path As String = FilePathText.Text.Substring(0, _
FilePathText.Text.LastIndexOf("\"))
        If System.IO.Directory.Exists(path) Then
            Try
                xmlSource.Save(FilePathText.Text)
                SaveResultsText.Text = FilePathText.Text & _
" was saved successfully."
            Catch saveErr As Exception
                SaveResultsText.Text = saveErr.ToString()
            End Try
        Else
            SaveResultsText.Text =
"Directory Does Not Exist, Try a different path."
            FilePathText.Text = ""
        End If

        Catch saveError As Exception
            SaveResultsText.Text = saveError.ToString()
        End Try
    End Sub

```

## Comments

Test the form using these steps:

1. For the XML source, use <http://aspalliance.com/cookbook/samples/cdcatalog.xml>.
2. Press the Load Xml Document button.
3. Now enter a valid path in your file system.
4. Press the Save button.
5. You will notice that the result of the save is outputted to the bottom TextBox.
6. Navigate to the path you gave in your file system to ensure the file was saved successfully.

### See Also

Section 12.1, "Opening an XML File"

## 12.4. Transforming an XML Document Using XSLT

You want to transform an XML document using XSLT.

### Technique

This example builds on Section 12.1 and enables the ASP.NET application to ask for a URL to an XSLT document, which it will then transform and output the result. You need to add a few new elements to the form. The first is a TextBox to allow the users to enter the URL of the XSLT document they want to use during transformation. Next, you need a literal control to hold the output of the XSLT transformation. Lastly, you'll add a Transform button, which will trigger the event handler to transform the XML and XSLT documents.

You need to create a method that will handle the transformation. The `Button` event handler will call this method.

The code is as follows:

```
Public Function Transform(ByVal docXml As System.Xml.XmlDocument, _
ByVal xslUrl As String, ByVal xslParamNames() As String, _
ByVal xslParamValues() As String) As String
    Dim xsltTransform As New System.Xml.Xsl.XsltTransform()
    Dim sResult As New System.IO.StringWriter()
    'load xsl to a xslttransform object
    Try
        xsltTransform.Load(xslUrl)
```

```

Catch exc As Exception
End Try

'load up the xsl parameters, if any
Dim xslArgs As New System.Xml.Xsl.XsltArgumentList()
If Not (xslParamNames Is Nothing) Then
    Dim counter As Integer = 0
    Dim paramname As String
    For Each paramname In xslParamNames
        xslArgs.AddParam(paramname, Nothing, xslParamValues(counter))
    Next paramname
End If

'try to transform it
Try
    ' call transform
    If Not (xslParamNames Is Nothing) Then
        xslTransform.Transform(docXml, xslArgs, sResult)
    Else
        xslTransform.Transform(docXml, Nothing, sResult)
    End If
Catch exc As Exception
End Try
Return sResult.ToString()
End Function 'Transform

```

Next, you call this method with the inputs from the ASP.NET Web Form with the Button event handler.

The Button event handler code is as follows:

```

If xmlSource Is Nothing Or xmlSource.InnerText = "" Then
    xmlSource.LoadXml(ResultText.Text)
End If
Try
    ResultsLiteral.Text = Transform(xmlSource, XslUrlTextBox.Text, _
        Nothing, Nothing)
    ResultsPanel.Visible = True
Catch transformError As Exception
    ResultsLiteral.Text = transformError.ToString()
End Try

```

## Comments

Test the form following these steps:

1. For the XML source, use <http://aspalliance.com/cookbook/samples/cdcatalog.xml>.

2. Press the Load Xml Document button.
3. For the XSLT source, use <http://aspalliance.com/cookbook/samples/cdcatalog.xsl>.
4. Press the Transform button.
5. The result of the transformation is displayed at the bottom LiteralControl.

### See Also

Section 12.1, "Opening an XML File"

## 12.5. Converting Between XML Documents and Datasets

You want to convert an XML document into a dataset, or convert a dataset into an XML document.

### Technique

The recipe is divided into three parts.

- Converting the XML document into a dataset
- Modifying data in the dataset
- Writing data in the dataset into an XML file

The work is done in the Page\_Load event handler:

```
Private Sub Page_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs)
    Try
        Response.Write("STEPS: <br>")
        Dim Ds As New DataSet
        ` Reading DSRead.XML file into DataSet Ds

        Ds.ReadXml(Server.MapPath("DSRead.XML"))
        Response.Write("1. DSRead.XML file is loaded
➔ into DataSet DS successfully.<br>")

        ` Modifying the data in Ds DataSet

        Ds.AcceptChanges()
        Ds.Tables(0).Rows(0).Item("LastModified") = DateTime.Now.ToString()
        Response.Write("2. Data is modified successfully
➔ in Ds DataSet.<br>")
    End Try
End Sub
```

```

        ' Writing Ds into XML DSWrite.XML file
        Dim dsChanges As DataSet
        dsChanges = Ds.GetChanges()
        dsChanges.WriteXml(Server.MapPath("") & "\DSWrite.xml")
        Response.Write("3. Ds DataSet changes are successfully
↳ written into DSWrite.XML file.<br>")
        Catch ex As Exception
            lblvalue.text = ex.Message
        End Try
    End Sub

```

## Comments

Working with simple XML files can be easy using ADO.NET's dataset. Its `ReadXml` and `WriteXml` methods allow quick conversion from XML to datasets and back again. Note that in order for this example to work, the users must have permission to write to the file specified.

### See Also

For the `DataSet.ReadXml` method—<http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemDataDataSetClassReadXmlTopic.asp>

For the `DataSet.WriteXML` method—<http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemDataDataSetClassWriteXmlTopic.asp>

Managing XML Data Using a Dataset in ADO .NET—

<http://www.msdnaa.net/Resources/Display.aspx?ResID=1489>

## 12.6. Creating a Class from an XML Document

You want to create a class from an XML document using the XSD.EXE tool, and use the created class in your Visual Studio .NET project.

### Technique

The XSD.EXE tool, which ships with the .NET Framework SDK, allows you to take an XML document and convert it into an XSD (an XML schema). After you have this XSD, you can use it along with the XSDI.EXE tool to generate a sub-classed dataset or a set of classes. This is handy when you only have the XML document, and do not want to parse it using XSLT.

This tool is a command line tool, so you must first dump to DOS. The easiest way in Visual Studio .NET to do this is to go to the Start menu, and then choose Programs, Microsoft Visual Studio .NET, Visual Studio .NET Tools. Then choose Visual Studio .NET Command Prompt.

If you do not have Visual Studio .NET, the location of the program on my machine is C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin\xsd.exe. If you do not have the path in the PATH environment variable, make sure you add that first.

Next take a look at the syntax of the command. Pay close attention to the /c and /d switches.

This example uses the XML document found at <http://aspalliance.com/cookbook/samples/cdcatalog.xml>.

First, issue the following command:

```
xsd.exe http://aspalliance.com/cookbook/samples/cdcatalog.xml
```

This should produce something similar to:

```
Microsoft (R) Xml Schemas/DataTypes support utility
[Microsoft (R) .NET Framework, Version 1.0.3705.0]
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.
```

```
Writing file 'C:\Inetpub\wwwroot\CookBook\Recipe1206\cdcatalog.xsd'.
```

Now that you have the XSD, you can generate a class file for this schema as follows:

```
xsd.exe cdcatalog.xsd /c
```

Produces:

```
Writing file 'C:\Inetpub\wwwroot\CookBook\Recipe1206\cdcatalog.cs'.
```

If you want to do this using VB.NET, simply add the /1:vb switch to the command.

Now that you have generated a class file, you can use it to load the XML document and then use that class file to manipulate the XML document. If you are not familiar with the XML serialization process, read up on it and continue reading after you feel comfortable with the topic.

The first thing you need to do is create a function that will load the XML document, and deserialize it into an instance of the catalog class. This project uses a new class created just to handle the serialization needs. Here it is:

```
Imports System
Imports System.Runtime.Serialization

Namespace Recipe1206vb
    Public Class Serialization
        Public Shared Function SerializeXML(ByVal request As Object, _
            ByVal type As System.Type) As System.IO.MemoryStream
            Try
                Dim serializer As New System.Xml.Serialization.XmlSerializer(type)
                Dim stm As New System.IO.MemoryStream()
                serializer.Serialize(stm, request)
                Return stm
            End Try
        End Function
    End Class
End Namespace
```

```

        Catch e As Exception
            Return Nothing
        End Try
    End Function 'SerializeXML

    Public Shared Function DeSerializeXML(ByVal envelope As String, _
ByVal type As System.Type) As Object
        Try
            Dim serializer As New System.Xml.Serialization.XmlSerializer(type)
            Dim stm As New System.IO.MemoryStream( _
System.Text.Encoding.ASCII.GetBytes(envelope))
            Dim ud As Object = serializer.Deserialize(stm)
            stm.Close()
            Return ud
        Catch e As Exception
            Return Nothing
        End Try
    End Function 'DeSerializeXML
End Class 'Serialization
End Namespace 'Recipe1206vb

```

Next you need to define the Catalog class:

```

<System.Xml.Serialization.XmlRootAttribute("catalog",
↳[Namespace]:="", IsNullable:=False)> _
Public Class catalog
    <System.Xml.Serialization.XmlElementAttribute("cd")> _
    Public Items() As catalogCD
End Class

Public Class catalogCD
    Public title As String
    Public artist As String
    Public country As String
    Public company As String
    Public price As String
    Public year As String
End Class

```

Now you can use that class to deserialize the catalog class:

```

Private Function LoadData(ByVal path As String) As catalog
    Try
        Dim fs As System.IO.FileStream = System.IO.File.OpenRead(path)
        Dim buff(fs.Length) As Byte
        fs.Read(buff, 0, CInt(fs.Length))
        fs.Close()
        cat = CType(Serialization.DeSerializeXML(
↳System.Text.Encoding.ASCII.GetString(buff), GetType(catalog)), catalog)
    End Try
End Function

```

```

    Return cat
Catch
End Try
End Function 'LoadData

```

The last major portion of the puzzle involves the process of saving the catalog class, with data that has been changed. This example includes enough functionality to enable you to edit any node and then press the Save button. This Save button consists of:

```

Private Sub SaveButton_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles SaveButton.Click
    Dim findvalue As String = titleDropDownList.SelectedItem.Text
    Dim foundedcd As catalogCD = Nothing
    Dim cd As catalogCD
    For Each cd In cat.Items
        If cd.title = findvalue Then
            foundedcd = cd
            Exit For
        End If
    Next cd
    If Not (foundedcd Is Nothing) Then
        foundedcd.artist = artistTextBox.Text
        foundedcd.country = countryTextBox.Text
        foundedcd.company = companyTextBox.Text
        foundedcd.price = priceTextBox.Text
        foundedcd.year = yearTextBox.Text
        Dim data As System.IO.MemoryStream = _
Serialization.SerializeXML(cat, GetType(catalog))
        Dim databytes As Byte() = data.ToArray()
        If System.IO.File.Exists(xmlPath) Then
            System.IO.File.Delete(xmlPath)
        End If
        Dim f As System.IO.FileStream = System.IO.File.OpenWrite(xmlPath)
        f.Write(databytes, 0, databytes.Length)
        f.Close()
    End If
End Sub

```

First you find the currently selected item from the catalog, and then set its members equal to the values provided on the form. Then, you take advantage of the serialization class again, this time to serialize the content into a `MemoryStream`. This enables you to easily save it using the `FileStream`, as shown previously. Note that this example requires the users to have write permission for the file being saved.

### See Also

XML Serialization—<http://msdn.microsoft.com/library/en-us/cpguide/html/cpconintroducingxmlserialization.asp>

## 12.7. Reading an XML Document Using an XmlTextReader

You want to read through an XML document using the `XmlTextReader` class.

### Technique

This example uses the `System.Xml` namespace. In `<script runat="server" />` block or codebehind:

```
Public Sub Page_Load(Source As Object, E As EventArgs)
    Dim _Reader As XmlTextReader
    Try
        _Reader = New XmlTextReader(Server.MapPath("users.xml"))
        Call ReadDocument(_Reader)
    Catch _Error As Exception
        ErrorLabel.Text = _Error.Message
    Finally
        _Reader.Close()
    End Try
End Sub
```

When the load event of the Web Form is raised, an object of the `XmlTextReader` class is created. This object is initiated with a reference to an XML document. The `Try/Catch` block is implemented to capture any errors that occur due to an improper reference or insufficient permissions.

```
Sub ReadDocument(ByRef _XmlReader As XmlTextReader)
    Dim sb As New System.Text.StringBuilder(100)
    Dim space As String = "&nbsp;"
    Dim space3 As String = "&nbsp;&nbsp;&nbsp;"

    While _XmlReader.Read()

        Select Case _XmlReader.NodeType

            Case XmlNodeType.Element
                sb.Append("<B>Element: </B>")
                sb.Append(_XmlReader.Name)
                sb.Append("<BR/>")

                'Print attributes for current node, if any available...
                If _XmlReader.AttributeCount > 0 Then
                    While _XmlReader.MoveToNextAttribute()
```

```

        sb.Append(space3)
        sb.Append("<B>Attribute Name: </B>")
        sb.Append(_XmlReader.Name)
        sb.Append(space)
        sb.Append("<B>Attribute Value: </B>")
        sb.Append(_XmlReader.Value)
        sb.Append("<BR/>")
    End While
End If

Case XmlNodeType.Text
    sb.Append(space3)
    sb.Append("<B>Value: </B>")
    sb.Append(_XmlReader.Value)
    sb.Append("<BR/>")

End Select

End While
OutputLiteral.Text = sb.ToString()
End Sub

```

The `ReadDocument()` method is responsible for printing the contents of the XML document. `Read()` property keeps the external while loop moving until the last node. Through the `NodeType` property, the current node type of the reader object is compared. If the current node happens to be an element, its name or attributes are displayed; if it happens to be text, the value of the text is displayed.

## Comments

The `XmlTextReader` class provides methods and properties to read through an XML document. However, you cannot navigate within the document freely. The object of the `XmlTextReader` class can only move forward. It works similar to the `Datareader` object in the .NET Framework. This means that after you have read a node you can only move forward, to the next node, and cannot move back unless you re-initialize the object (starting at the beginning).

To simply reformat some XML, XSLT provides a much cleaner and more efficient method.

### See Also

Section 12.8, "Writing an XML Document Using an `XmlTextWriter`"

MSDN—search for `System.Xml.XmlTextReader`

## 12.8. Writing an XML Document Using an XmlTextWriter

You want to write an XML document using the `XmlTextWriter` class.

### Technique

Import `System.Xml` namespace for this example. In `<script runat="server" />` block or codebehind:

```
Public Sub Page_Load(Source As Object, E As EventArgs)
    Dim _Writer As New XmlTextWriter(Server.MapPath("users.xml"), Nothing)
    'Nothing means use default UTF-8 format

    'Default formatting property is None
    _Writer.Formatting = Formatting.Indented

    Try
        _Writer.WriteStartDocument(True)
        _Writer.WriteStartElement("users")

        Call WriteUsers(_Writer)

        _Writer.WriteEndElement()
        MessageLabel.Text = "File successfully created..."

    Catch _Error As Exception
        MessageLabel.Text = _Error.Message

    Finally
        _Writer.Flush()
        _Writer.Close()
    End Try
End Sub
```

Next, set the formatting property for the writer object. This will write the elements in the target file in an indented format. The `WriteStartDocument()` method writes the first line of the XML document, and encloses the XML document declaration within the `<?>` delimiters. The `WriteStartElement()` writes an element of the specified name that's provided to it. In order to write the closing tag of an element, its name need not be specified. The object itself checks for the last opened tag and automatically writes the closing tag for the related element. The `Flush()` and `Close()` calls ensure that the XML is correctly written to the file and the output stream is closed after the task is completed.

```
Public Sub WriteUsers(ByRef _output As XmlTextWriter)
```

```
With _output

    'First user...
    .WriteStartElement("user")
    .WriteAttributeString("role", "admin")

    .WriteStartElement("username")
    .WriteString("jsmith")
    .WriteEndElement()
    .WriteStartElement("password")
    .WriteString("john")
    .WriteEndElement()

    .WriteEndElement()

    'Second user...
    .WriteStartElement("user")
    .WriteAttributeString("role", "operator")

    .WriteStartElement("username")
    .WriteString("tcruise")
    .WriteEndElement()
    .WriteStartElement("password")
    .WriteString("tom")
    .WriteEndElement()

    .WriteEndElement()

End With
```

```
End Sub
```

This method writes the user data into the XML file. The `writeUsers` method is passed a reference to the `XmlTextWriter` object. The various methods of the object are then called to write elements, attributes, and their values to the XML document being produced. The `writeAttributeString()` accepts two arguments. The first is the name of the attribute itself and the second is the value of the attribute. The `writeString()` method accepts a single argument of string type. This method is used for writing the content of an element. Here, you can also see that the `writeEndElement` is called without being passed any parameters. This is because the object checks out the last opened element tag and writes the appropriate closing element tag.

## Comments

The `XmlTextWriter` class provides a serialized, forward-only way to write XML to a file. The class exposes various methods and properties to create components of an XML



```

sb.Append("<BR/>")

'Move to first child element :: User...
_Nav.MoveToFirstChild()

Do
    sb.Append(space2)
    sb.Append("name=" & _Nav.Name & ", type=" & _
_Nav.NodeType.ToString())
    sb.Append("<BR/>")

    'Get Attributes...
    _Nav.MoveToFirstAttribute
    sb.Append(space2)
    sb.Append("Attribute: " & _Nav.Name & "=" & _Nav.Value)
    sb.Append("<BR/>")

    'Bring the navigator back to attribute parent
    _Nav.MoveToParent()

    'Move to first child element :: Username...
    _Nav.MoveToFirstChild()

Do
    sb.Append(space3)
    sb.Append("name=" & _Nav.Name & ", type=" & _
_Nav.NodeType.ToString() & ", value=" & _Nav.Value)
    sb.Append("<BR/>")
    Loop While _Nav.MoveNext()

    _Nav.MoveToParent()

Loop While _Nav.MoveNext()

OutputLiteral.Text = sb.ToString()

End Sub

```

## Comments

The `XPathNavigator` object cannot be created directly. You must first create the `XmlDocument` or `XPathDocument` in order to create one. These classes provide a `CreateNavigator()` method that returns an `XPathNavigator`. The `XPathNavigator` class exposes methods for navigating within an XML document. Some of these methods are `MoveToNext()`, `MoveToPrevious()`, `MoveToRoot()`, `MoveToFirstChild()`, and `MoveToParent()`. Using the `XPathNavigator` to navigate XML documents is fairly

straightforward. All you have to do is call a property or a method of the class to move to a particular node in the document or display a node's name, type, value, and so on, as demonstrated in the previous example.

**See Also**

MSDN Library—[System.Xml.XPath.XPathNavigator](#)