An Introduction to XML Processing with Visual Basic 2005 (8.0)

Lars Knösel

January 22, 2008

XML - Introduction

XML...

- stands for Extensible Markup Language
- was designed to describe data
- -tags are not predefined
- is **not** HTML
- is going to be everywhere

Advantages of XML

- XML documents use a self-describing and simple syntax
- The rules to build an XML document are very easy to learn
- Creating software that process XML data is quite easy
- Converting data to XML can greatly reduce the trouble of exchanging complex data
- The usage of plain text files for exchanging XML documents is a big advantage

Structure of an XML Document I

- Declaration
- Elements
- Attributes
- Comments
- CDATA sections
- (Processing Instructions)

Structure of an XML Document II

XML file example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<summits>
  <summit>
    <name>Tonigenkogel</name>
    <height>3013</height>
    <!-- Comment. -->
    <grade>I</grade>
    <region>Stubaier Alpen</region>
    <date>2007-08-03</date>
    <note>Nordwestgrat</note>
  </summit>
  <summit>
    ...
</summits>
```

Important Classes in Namespace System.Xml

- XmlDocument
- XmlElement
- XmlException
- XmlNode
- XmlNodeList
- XmlReader
- XmlWriter

Important Members of Class XmlDocument

- Public Overridable Sub Load(ByVal file As String)
- Public Overridable Sub Save(ByVal file As String)
- Public Overridable Function CreateNode(ByVal nodeTypeString As String, ByVal name As String, ByVal namespaceURI As String) As System.Xml.XmlNode
- Public Function CreateElement(ByVal name As String) As System.Xml.XmlElement

Important Members of Class XmlNode

- Public Overridable Property InnerText() As String
- Public Overridable Function AppendChild(ByVal newChild As System.Xml.XmlNode) As System.Xml.XmlNode
- Public Function SelectNodes(ByVal xpath As String) As System.Xml.XmlNodeList
- Public Function SelectSingleNode(ByVal xpath As String) As System.Xml.XmlNode
- Public Overridable Function RemoveChild(ByVal oldChild As System.Xml.XmlNode) As System.Xml.XmlNode

Some XmlNode Types

- Element
- Attribute
- Text
- CDATA
- Comment
- XmlDeclaration

Reading an XML Document

Reading an XML document is a piece of cake:

```
Imports System.Xml 'Outside of module!

Dim doc As New XmlDocument()

Sub read()
  Try
    doc.Load("summits.xml")
    Dim summitNames As XmlNodeList = _
      doc.SelectNodes("//summits/summit/name")
    For Each name As XmlNode In summitNames
      Console.WriteLine(name.InnerText())
    Next
  Catch ex As Exception
  ...
End Sub
```

XPath Examples

- Select all summit-Nodes:

```
//summits/summit
```

- Select a summit with a special name:

```
//summits/summit[name='Similaun']
```

- Select all summits whose name starts with 'H':

```
//summits/summit/name[starts-with(.,'H')]
```

- Select all summits whose height is lower than 2500 meters:

```
//summits/summit[height < 2500]
```

Adding a Node

```
Dim doc As New XmlDocument()

Sub addNode()
  Dim root, node, summit As XmlNode
  Try
    doc.Load("summits.xml")
    root = doc.SelectSingleNode("//summits")
    summit = doc.CreateNode(XmlNodeType.Element, _
      "summit", Nothing)
    node = doc.CreateNode(XmlNodeType.Element, _
      "name", Nothing)
    node.InnerText = "Foobar"
    summit.AppendChild(node)
    root.AppendChild(summit)
    doc.Save("summits.xml")
  Catch ex As Exception
  ...
```

Editing a Node

```
Dim doc As New XmlDocument()

Sub editNode(ByRef name As String, _
  ByRef element As String, ByRef value As String)
  Try
    doc.Load("summits.xml")
    Dim summit As XmlNode = _
      doc.SelectSingleNode("//summits/summit[name='" _
      & name & "']")
    summit.SelectSingleNode(element).InnerText = value
    doc.Save("summits.xml")
  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Deleting a Node

```
Dim doc As New XmlDocument()

Sub deleteNode(ByRef name As String)
  Dim root, summit As XmlNode
  Try
    doc.Load("summits.xml")
    root = doc.SelectSingleNode("//summits")
    summit = _
      root.SelectSingleNode("//summits/summit[name='" _
      & name & "']")
    root.RemoveChild(summit)
    doc.Save("summits.xml")
  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Digression Into XSL(T)

- XSL is an extensible stylesheet language family (XSL-FO, XSLT, XPath)
- XSL-FO is a language for expressing stylesheets
- XSLT is a language for transforming XML
- Stylesheet processor (tree transformation and formatting)

Important XSLT-Elements

- xsl:choose
- xsl:copy
- xsl:for-each
- xsl:if
- xsl:sort
- xsl:stylesheet
- xsl:template
- xsl:value-of

A Transformation Stylesheet I

Insertion of a unique id into each summit node:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" />

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
  <!-- Continued on next slide. -->
```

A Transformation Stylesheet II

Insertion of a unique id into each summit node:

```
  <xsl:template match="summit">
    <xsl:copy>
      <xsl:attribute name="uid">
        <xsl:value-of select="generate-id(.)" />
      </xsl:attribute>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

A Transformation Stylesheet III

Transformation into HTML

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<xsl:stylesheet  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" />

  <xsl:template match="/">

  <!-- Some html code goes here. -->
```

A Transformation Stylesheet IV

Transformation into HTML

```
    <xsl:for-each select="summits/summit">
      <tr>
        <td><xsl:number format="1" /></td>
        <td><xsl:value-of select="name" /></td>
        <td><xsl:value-of select="height" /></td>
        <td><xsl:value-of select="grade" /></td>
        <td><xsl:value-of select="region" /></td>
        <td><xsl:value-of select="date" /></td>
        <td><xsl:value-of select="note" /></td>
      </tr>
    </xsl:for-each>
    <!-- Some html code goes here. -->
  </xsl:template>
</xsl:stylesheet>
```

Performing an XSL Transformation

```
Dim doc As New XmlDocument()
Dim xslt As New Xsl.XslCompiledTransform
Dim writer As XmlWriter
Dim wtSettings As New XmlWriterSettings

Sub transform()
  wtSettings.Encoding = Text.Encoding.GetEncoding("iso-8859-15")
  wtSettings.Indent = True
  wtSettings.OmitXmlDeclaration = True
  Try
    writer = XmlWriter.Create("summits.html", wtSettings)
    xslt.Load("transform_html.xsl")
    doc.Load("summits.xml")
    xslt.Transform(doc, writer)
  Catch ex As Exception
  ...
End Sub
```

Digression Into XML Schema

XML Schemas...

- are an alternative to DTDs
- describe the structure of an XML document
- are richer and more powerful than DTDs
- use XML syntax
- support data types
- support namespaces
- are extensible

Important XSD Data Types

- xs:boolean
- xs:date
- xs:decimal
- xs:integer
- xs:string
- xs:time

An XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="summits">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="summit">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="height" type="xs:integer"/>
              <xs:element name="grade" type="xs:string"/>
              <xs:element name="region" type="xs:string"/>
              <xs:element name="date" type="xs:date"/>
              <xs:element name="note" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        ...
```

Validating XML Against a Schema I

```
Dim doc As New XmlDocument()
Dim reader As XmlReader
Dim rdSettings As New XmlReaderSettings()

Sub validate()
  rdSettings.ValidationType = ValidationType.Schema
  AddHandler rdSettings.ValidationEventHandler, _
    AddressOf onError
  Try
    rdSettings.Schemas.Add(Nothing, "summits.xsd")
    reader = XmlReader.Create("summits.xml", rdSettings)
    doc.Load(reader)
  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Validating XML Against a Schema II

```
Sub onError(ByVal sender As Object, _
  ByVal ex As Schema.ValidationEventArgs)
  Console.WriteLine(ex.Message)
End Sub
```

References and Further Reading I

- Monadjemi, Peter: Visual Basic 2005: Windows-Programmierung mit Visual Basic 2005, Visual Studio 2005 und dem .NET Framework 2.0, München: Markt+Technik, 2006
- Bierhaus, Alex/Kotz, Jürgen: Visual Basic 2005: Einstieg für Anspruchsvolle, München: Pearson Studium, 2006
- Stein, Markus: workshop XML, München: Addison-Wesley, 2002
- http://www.w3.org/XML/ [2007-12-28]
- http://www.w3.org/Style/XSL/ [2008-01-20]
- http://www.w3.org/XML/Schema/ [2008-01-20]

References and Further Reading II

- http://www.w3schools.com/xml/ [2007-12-28]
- http://www.w3schools.com/xsl/ [2008-01-20]
- http://www.w3schools.com/schema/ [2008-01-20]

Thanks!