Table of Contents: XML Development using ASP.NET

XML in the .NET Platform

- What is the .NET Platform?
- What is an assembly?
- What can I do with the System.Xml assembly?
- What type of APIs does the System.Xml assembly support?
- A few System.Xml assembly classes:
  - XmlDocument
  - XmlElement
  - XmlNode
  - XmlNamedNodeMap
  - XmlNodeList
  - XmlTextReader
  - XmlTextWriter

What is the .NET Platform?

- .NET is Microsoft's new platform for developing and delivering web-based and/or client-server applications.
- The .NET platform offers many advantages over "classic" ASP/XML programming:
  - XML is integrated directly into the .NET platform.
  - ADO.NET makes it easy to switch between XML and relational data views.
  - The .NET platform supports distributed computing through using Web Services.
  - ASP.NET code is compiled offering better performance.
  - Cross-language support allows VB.NET components to be used by C# and many other .NET languages.
  - Too many new features to mention!

What is an Assembly?

- The .NET documentation defines an assembly in the following manner:
  - An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality, a “logical” dll.
- In sum, an assembly takes several physical files such as interfaces, classes, resource files, etc. and creates metadata referred to as a manifest about how the files work together.

What can I do with the System.Xml Assembly?

- **The .NET platform was built to support XML from the ground up. As a result, the System.Xml assembly allows the following types of functionality to be integrated into .NET applications:**
  - **XML 1.0 Standard - including DTD support (XmlTextReader)**
  - **XML Namespaces - both stream level and DOM.**
  - **XML Schemas supported for schema mapping and serialization.**
  - **DOM Level 2 Core (XmlDocument)**
  - **SOAP 1.1 (including the Soap Contract Language and Soap Discovery)**
  - **XPath expressions**
  - **XSL/T transformations (XslTransform)**
  - **Forward-Only Cursor Model**

What type of APIs does the System.Xml assembly support?

- XML documents can be parsed using one of the following APIs:
  - Forward-Only Cursor Model (XmlTextReader)
  - Document Object Model - DOM (XmlDocument)
- Each of these mechanisms will be discussed in the following sections.

The XmlTextReader Class

- The XmlTextReader class provides a fast and memory efficient way to parse an XML document. This is accomplished by treating the XML document as a stream.

- Although this type of functionality has its limitations (read-only), it provides an excellent mechanism for working with large XML documents.

- The XmlTextReader class exposes a pull model as compared to the push model found in the Simple API for XML (SAX).

Pull Vs. Push

- SAX is a popular way to work with larger XML documents in a fast and efficient manner.

- SAX is based upon a push model. This model works by pushing information about nodes found within an XML document to a ContentHandler class.

- The push model found in SAX is NOT explicitly supported by the System.Xml assembly. Instead, the XmlTextReader class uses a pull model.

Advantages of the Pull Model

- **The pull model has the following advantages over the push model found in SAX:**

  - **State Management - Push model content handlers must build very complex state machines that a pull model client can greatly simplify by simply managing the state by natural top-down procedural refinement.**

  - **Multiple Input Streams - A pull model allows a client to splice together multiple input streams. Doing this with a push model can prove to be difficult.**

Advantages of the Pull Model (Continued)

- **Layering Test - A push model can easily be built on top of a pull model, while the reverse is not true. A SAX implementation written using the Pull model found in .NET can be downloaded from: http://www.XMLforASP.NET.**

- **Hints from client - A pull model API can be designed to allow the client to give hints to the parser about what they are expecting next. This allows the parser to optimize for that. For example, in data type support, when a client knows the next item to process is supposed to be an integer, the parser can parse the integer right out of the parser buffer instead of returning a string which is subsequently thrown away.**

Advantages of the Pull Model (Continued)

- **The pull model has the following advantages over the push model found in SAX:**

  - **Avoids Extra Copy - A pull model allows the client to give the parser the buffer into which to write the strings. This avoids the extra copy from the parser buffer to the string object which is then pushed to the client buffer.**

  - **Skipping Things - The push model has to push everything including all the attributes, comments, text, whitespace, etc. With a pull model, the client pulls only what they are interested in. If, for example, the client doesn't read the attributes then all those attribute values do not need to be entity expanded, values "stringized", names atomized, etc. This model allows for more efficient messaging level applications of XML.**

Using the XmlTextReader Class

The XmlTextReader is very simple to instantiate and use:

```
using System.Xml;
public class ReadXmlFile       {
   StringBuilder output = new StringBuilder();

   public string ReadDoc(String doc) {
      XmlTextReader xmlReader = null;
      try  {
         xmlReader = new XmlTextReader(doc);
         WriteXml(xmlReader);
      }
      catch (Exception e) {
         output.Append("Error Occured While Reading " +
            doc + " " + e.ToString());
      }
      finally {
         if (xmlReader != null)
            xmlReader.Close();
      }
      return output.ToString();
   }
}
```

Using the XmlTextWriter Class

- The XmlTextReader class is complimented by the XmlTextWriter class

- The XmlTextWriter class performs the task of writing to an XML document in a forward-only/cursor-style manner.

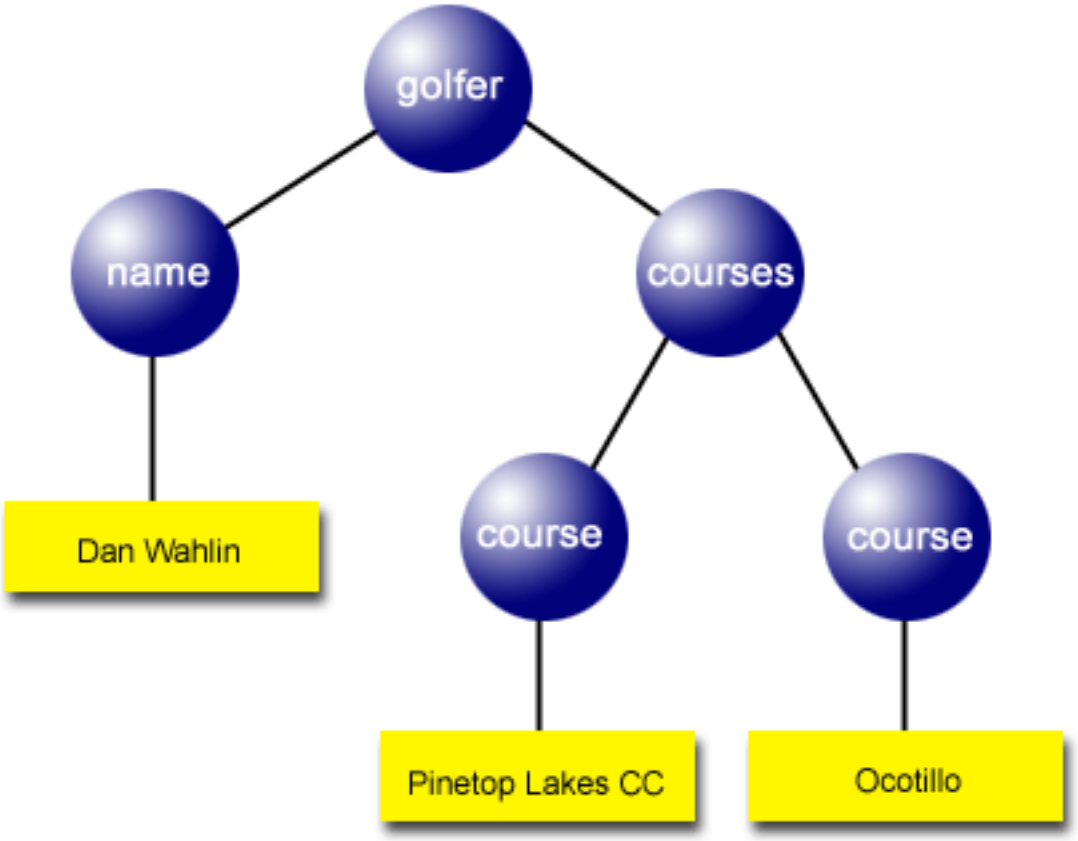
Using the XmlTextWriter Class (Continued)

- **Using the XmlTextWriter is very simple once you familiarize yourself with its properties and methods:**

```
<%@ Import Namespace="System.Xml" %>
<script language="C#" runat="server">
public void Page_Load(Object Src, EventArgs E) {
      string xmlDoc = Server.MapPath("xmltextwriter.xml");
      XmlTextWriter writer = null;
      try   {
        writer = new XmlTextWriter(xmlDoc,Encoding.UTF8);
        writer.Formatting = Formatting.Indented;
        writer.WriteStartDocument(true);
        writer.WriteComment("XML Nodes added using the XmlTextWriter");
        writer.WriteStartElement("golfers");
          writer.WriteStartElement("golfer", null);
            writer.WriteAttributeString("skill","moderate");
            writer.WriteAttributeString("handicap","12");
            writer.WriteAttributeString("clubs","Taylor Made");
            writer.WriteAttributeString("id","1111");
           writer.WriteEndElement(); //golfer
        writer.WriteEndElement(); //golfers
        //.....More Code Follows
}
</script>
```

Using the XmlDocument Class to work with the DOM

- **The Document Object Model (DOM) places each piece of information within an XML document into a tree structure:**



- **Each section of the tree is referred to as a "node" and can be updated, inserted, deleted, or moved.**

Using the XmlDocument Class to Add Nodes

Loading an XML document into the DOM and adding nodes:

```
<%@ Import Namespace="System.Xml" %>
 <script language="C#" runat="Server">
  void Page_Load(object sender, EventArgs e) {
    XmlDocument oDocument = new XmlDocument();
    oDocument.Load(Server.MapPath("xmlDocument.xml"));
    XmlNode oRoot = oDocument.DocumentElement;
    try {
       XmlElement oElement1 = oDocument.CreateElement("testB");
       XmlElement oElement2 = oDocument.CreateElement("testC");
       oElement2.SetAttribute("myAtt","myAttValue");
       oRoot.AppendChild(oElement1);
       oRoot.AppendChild(oElement2);
    }
    catch(Exception exc) {
       Response.Write(exc.ToString());
    }
    Response.ContentType ="text/xml";
    oDocument.Save(Response.Output);
  }
 </script>
```

The XmlNodeList Class

- The XmlNodeList class represents a collection of nodes

- Enumerating through a collection of nodes:

```
<%@ Import Namespace="System.Xml" %>
<script language="C#" runat="Server">
  void Page_Load(object sender, EventArgs e) {
     XmlDocument xmlDoc = new XmlDocument();
     xmlDoc.Load(Server.MapPath("xmlDocument.xml"));
     XmlNode root = xmlDoc.DocumentElement;

     XmlNodeList oNodeList = root.ChildNodes;
     foreach (XmlNode oCurrentNode in oNodeList) {
        Response.Write(oCurrentNode.Name + " ");
        if (oCurrentNode.HasChildNodes) {
            Response.Write("(Children: " +
              oCurrentNode.ChildNodes.Count + ")");
        }
     }
  }
</script>
```

The XmlNamedNodeMap Class

- **The XmlNamedNodeMap class represents a collection of attributes for a given element node:**

```
<%@ Import Namespace="System.Xml" %>
<script language="C#" runat="Server">
  void Page_Load(object sender, EventArgs e) {
      StringBuilder output = new StringBuilder();
      XmlDocument xmlDoc = new XmlDocument();
      xmlDoc.Load(Server.MapPath("xmlNamedNodeMap.xml"));
      XmlNode oChild = xmlDoc.DocumentElement.FirstChild;
      if (oChild.Attributes.Count > 0) {
         XmlNamedNodeMap oNamedNodeMap = oChild.Attributes;
         output.Append("<b>" + oChild .Name + "</b> ");
         foreach (XmlAttribute att in oNamedNodeMap) {
               output.Append(" <i>" + att.Name + "</i>=\"" +
                 att.Value + "\" ");
         }
      }
      Response.Write(output.ToString());
   }
</script>
```

Summary

- The .NET platform was built to support XML from the ground up.
- The System.Xml assembly offers many different classes that allow you to integrate XML directly into your .NET applications.
- For more information on Dan Wahlin's hands-on 3 day course titled "XML for ASP.NET Developers" visit:
  http://www.XMLforASP.NET