Cst556 Distributed Applications for .NET with Mono

3. XML and SOAP
3.a Outcomes and Reference for XML and SOAP?

3.a.1 Motivation

- What is XML, what are some of the derivative XML standards, what is the importance of XML to distributed computing, how can I process XML in a program, and what tools are available to process XML for me?

- **Section Outcomes**
  - To know the format of an XML document and an XML schema. To understand the type of information that can be described in each and how this relates to distributed web computing.
  - To understand how to specify an XML document and corresponding XML schema.
  - To know how to process an XML document with a C# .NET program.
  - To know how to process an XML Schema with a C# .NET program.
  - To know what a DOM (document object model) tree is; to know how to create and use the information in a DOM tree for a simple XML document.
  - To understand the use and form of an XML stylesheet and its corresponding language.
3.a.2 Index of Example Programs on XML

- Xml, Schema, and DOM reading and writing: Section 3.c.7 Example XML
- XSLT Conversions: Section 3.e.6 XSLT: Example Conversion
- TreeView Controls: Section 3.g.3 Manipulating TreeView
3.b Introduction to XML

3.b.1 Why is XML Important?

- Distributed applications need to communicate data.

- **Data Interoperability**
  - **Answers**: How much work to make one tool’s data useful to another tool?
  - Look at your data files. How many are useful to more than a single application (tool) on your system?
  - **What format** is used for such files? Most likely its **text-based**, especially if the tools are produced by different vendors.

- **Interoperability** can be obtained in two ways:
  - Tools use a **common internal structure** - e.g. Microsoft CLR requires a common format for (compiler generated) object code.
  - Dynamic conversion - **Filters** allow Microsoft Word documents to be opened in Adobe FrameMaker.

- Use of **XML** is widespread
  - XML-RPC, SOAP, WSDL, UDDI
  - XML is a **text-based** language specifying **semantics (meaning)** of data
3.b.2 Why is XML Important with .NET?

• As one example, that is important to distributed web-based applications, .NET and web services must interface relational databases.
  - ADO.NET defines that interface
  - The class that is primary to the interface is `System.Data.DataSet`

• What about the class `System.Data.DataSet`?
  - In-memory cache of data retrieved from or bound to a database
  - A `DataSet` is a collection of related `DataTable` objects reflecting the database contents.

• How can data from a database be transported over the network?
  - `DataSet` includes methods to read and write information to and from XML and XML schema

• `DataSet`
  - A `DataSet` reads and writes data and schema as XML documents
  - XML data and schema can be transported across HTTP and used by any application that can process XML.
3.b.3 What is XML?

- Extensible Markup Language (XML) 1.0 (2nd edition 6 October 2000)
  - see: http://www.w3.org/XML/
- XML: text-based mark-up language to specify data semantics (meaning).
  - Tags identify characteristics of data, for example a name tag
  - `<name>Tim Lindquist</name>`
  - Tags are identifiers in angle brackets, with opening and closing definition.
  - Multiple tags characterizing data are called mark-up.
- HTML: a similar mark-up language specifying how to display information
  - `<h2>What is XML</h2>`
  - Indicates What is XML is to be rendered as a level 2 heading
- XML tags identify data
  - `<model>Audi 5000</model>`
  - The string Audi 5000 specifies a (car) model.
  - XML element: A tag (from beginning to end) together with the data enclosed within the tag define an element of XML data.
3.b.4 XML Elements and Attributes

- Often what can be done without attributes, such as:

```xml
<faculty>
    <name>Tim Lindquist</name>
    <email>Tim.Lindquist@asu.edu</email>
    <course>Cst556 Distributed Applications for Windows</course>
    <course>Ser321 Principles of Distributed Software Systems</course>
    <course>Ser502 Emerging Programming Languages</course>
</faculty>
```

- Here, **name** and **email** are elements nested in the **faculty** element.

- Can also be done with attributes, such as:

```xml
<faculty name="Tim Lindquist" email="Tim@asu.edu">
    <course>Cst556 Distributed Applications for Windows</course>
    <course>Ser321 Principles of Distributed Software Systems</course>
    <course>Ser502 Emerging Programming Languages</course>
</faculty>
```

- Here, **name** and **email** are attributes of the **faculty** element.

- XML must be **well-formed**
  - Multiple attributes are separated by spaces
  - All tags must have a closing: `<myTag ... />` or `<myTag> ... </myTag>`
  - Elements are properly nested - **not** `<myTag>...<tag>...<myTag>...</tag>`
  - see: [http://www.ucc.ie/xml/#FAQ-VALIDWF](http://www.ucc.ie/xml/#FAQ-VALIDWF)
3.b.5 Other XML Elements

• Comments in XML
  - Information generally not processed <!-- a comment -->, see example below

• Flags in XML provide a data-less information marker for the XML.

  <faculty name="Tim Lindquist" email="Tim@asu.edu">
    <!-- a flag tag indicates a property of this faculty; eg, deptChair -->
    <departmentChair/>
    <course>Cst591 Computing Studies Graduate Seminar</course>
  </faculty>

• XML prolog - Each XML file starts with a prolog.
  - <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
  - Version is required.
  - Encoding - identifies the character set used in the specification. The default is compressed Unicode: UTF-8; Western Europe and English use Latin-1, which is ISO-8859-1.
  - Standalone tells whether the document references external documents.
3.c XML Schema

3.c.1 What is a Schema and Why use them?

• Deciding which approach (element versus attribute) is a schema decision.
• XML Schema provides a mechanism to define the syntax and other constraints of an XML specification. In a schema, you define what’s legal in your XML documents.
• Schema define:
  - what elements are used in a specification and their names
  - the type of an element - complex or string, for example
  - any attributes associated with an element and its properties
  - what nesting and order is permitted within elements.
• See the on-line www.w3 schema tutorial at:
  - https://www.w3schools.com/xml/schema_intro.asp
• There is also a Free Software Foundation Tutorial available at:
  - This is a set of powerpoints and examples, including a Java validator.
3.c.2 XML Schema Makeup

- **xsd:schema** element encloses all definitions of an Xml schema and defines aspects of the namespace.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://pooh.poly.asu.edu/Cst556/Examples/schema"
    xmlns="http://pooh.poly.asu.edu/Cst556/Examples/schema"
    elementFormDefault="qualified">
    <!-- the rest of the schema definition -->
</xsd:schema>
```

- **xmlns:xsd** - gives the namespace for finding all elements and attributes used in this schema specification (element, schema, complexType, sequence, integer, boolean, string)

- **targetNamespace** attribute gives the namespace in which all identifiers defined in the schema will reside.

- **xmlns** - gives the default namespace that is used to qualify identifier references. For example, **ref faculty** would be resolved to the faculty identifier in **http://pooh.poly.asu.edu/Cst556/Examples/schema**

- **elementFormDefault** - indicates that any instance documents which conform to this schema must qualify the namespace of elements referenced.
3.c.3 Defining Elements in a Schema Definition

- Elements are defined in a schema to be either simple or complex
  - Simple elements have only text (e.g., date, string, integer, boolean)
  - `<price>25.75</price>`
  - Complex elements may have attributes, nested elements and/or text.
  - `<movie category="Bond">Tomorrow Never Dies</movie>`

- Simple elements have no attributes or nested elements, but may have a type.
  - `<xs:element name="elementName" type="elementType"/>`
  - For example, `<xs:element name="price" type="xs:decimal"/>`

- The simplest form of complex elements combines text and attributes
  - The `movie` element above, defined with its own reusable `movieType`.

```xml
<xs:element name="movie" type="movieType"/>
<xs:attribute name="category" type="xs:string"/>
<xs:complexType name="movieType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="category"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```
3.c.4 More on Complex Element Types

• Complex elements are defined with indicators controlling nested elements.

• **Order indicators** specify ordering of directly nested elements
  - `<sequence>` Child elements must appear in the same order as declared.
  - `<all>` Child elements appear in any order and all appear exactly once.
  - `<choice>` Specifies that one of the elements can occur

• **Occurrence indicators** (*minOccurs* and *maxOccurs*) are attributes specifying how many times an element may occur.

```xml
<xs:element name="breed" type="xs:string"/>
<xs:element name="age" type="xs:positiveInteger"/>
<xs:element name="trick" type="xs:string"/>
<xs:attribute name="name" type="xs:string"/>
<xs:element name="cat">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="breed" minOccurs="0" maxOccurs="2"/>
      <xs:element ref="age" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="trick" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="name"/>
  </xs:complexType>
</xs:element>
```
3.c.5 Attributed Elements

- Complex schema elements can have attributes.
  - `<xs:attribute name="myAttr" type="xs:string" />`

- Some of the types are:
  - `xs:string`
  - `xs:decimal`
  - `xs:integer`
  - `xs:boolean`
  - `xs:date`
  - `xs:time`

- Or, there may be a user defined type.

- Other attributes available for `xs:attribute` elements are:
  - `default="MyString"` -- provides a default value for the attribute
  - `fixed="MyString"` -- won’t let the attribute value change from MyString
  - `use="required"` or `use="optional"` -- optional is the default if use is not specified.
3.c.6 Example Schema and XML

- Schema to define a faculty in a department: `faculty.xsd`.

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="tel:faculty-schema" xmlns="tel:faculty-schema"
    elementFormDefault="qualified">
    <xsd:element name="department">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="faculty" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

<!-- to get validation and DOM to recognize attributes, they are defined at the
outer level and redefined within the appropriate complexType. -->
```xml
<xsd:attribute name="name" type="xsd:string" />
<xsd:attribute name="email" type="xsd:string" />
<xsd:element name="title" type="xsd:string" />
<xsd:element name="employYear" type="xsd:integer" />
<xsd:element name="faculty">
    <xsd:complexType>
        <xsd:sequence maxOccurs="unbounded">
            <xsd:element ref="title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="employYear" minOccurs="1" maxOccurs="1" />  
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="email" type="xsd:string" />
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```
3.c.7 Example XML

- Here is an example xml document using the schema. See: faculty.xml

```xml
<?xml version="1.0"?>
<department xmlns="tel:faculty-schema">
    <faculty name="Tim Lindquist" email="Tim@asu.edu">
        <title>Professor and Chair</title>
        <employYear>1985</employYear>
    </faculty>
    <faculty name="Ima Faculty" email="Ima@asu.edu">
        <title>A Great Professor</title>
        <employYear>1990</employYear>
    </faculty>
</department>
```

- See the sample program to read in the schema definition
  - This example and the other Xml examples with and Ant build file:
  - see the program ReadASchema in the project: xmlEgs.jar
3.d  Manipulating XML with C#

3.d.1  Reading and Writing Xml Documents with C#

- .NET supports two approaches for manipulating Xml documents with C#
  - `XmlTextReader` and `XmlTextWriter` classes provide methods and properties to read and write streams of xml.
  - `XmlDocument` provides methods and properties that allow you to read an xml document (creating an internal hierarchy of nodes), modify the structure and write it back out. (Document Object Model - DOM).

- The `XmlTextReader` and `XmlTextWriter` classes efficiently read and write XML.
  - Use these classes when you do not need to manipulate an internal document structure.
  - You do not get a DOM with `XmlReader` and `XmlWriter`.
  - These provide non-cached forward only access to streams of Xml
  - See `XmlTextReaderExample` and `XmlTextWriterExample` in: `xmlEgs.jar`
3.d.2 Document Object Model

- Example XML from Microsoft .NET documentation

```xml
<?xml version="1.0"?>
<books>
  <book>
    <author>Carson</author>
    <price format="dollar">31.95</price>
    <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
    <publisher>MSPress</publisher>
    <state>WA</state>
  </pubinfo>
</books>
```

- The document object model for this XML is:
3.d.3 C# .NET Classes to Support DOM

- Here is an example using `XmlDocument` directly from .NET documentation

```csharp
XmlDocument doc = new XmlDocument();
//Load the the document with the last book node.
XmlNodeReader reader = new XmlTextReader("books.xml");
reader.WhitespaceHandling = WhitespaceHandling.None;
reader.MoveToContent();
reader.Read();
reader.Skip(); //Skip the first book.
reader.Skip(); //Skip the second book.
doc.Load(reader);
doc.Save(Console.Out);
```

- Exercise, construct a program with the above code
- Run the program with the file (you may need to view source to see this file):
  - `books.xml`
  - Modify the program to read in all books and to add a new book by calling `Load()` or `LoadXml` method in `XmlDocument`.
3.d.4  Xml Document Hierarchy XmlDocument

- After initialized, the XmlDocument object is a tree of Nodes that can be manipulated under program control.

- Traversing the tree
  - The tree is made up of XmlNode objects. The XmlDocument has a list of xml nodes, accessed via the HasChildNodes and ChildNodes properties.

```csharp
foreach (XmlNode node in myXmlDoc.ChildNodes)
{
    Console.WriteLine("Node type is: {0}, node.NodeType");
}
```

- Each XmlNode object also has a List of child nodes accessed in the same way.

- The Attributes property of an XmlNode is an XmlAttributeCollection, which is a potentially null collection of XmlAttribute objects.

```csharp
if (myXmlDoc.Attributes != null) {
    Console.WriteLine("value of Name attribute is: {0}", myXmlDocAttributes["Name"]);
}
```

- XmlDocument also has methods for modifying the XmlDocument.

- An example that reads the books.xml file into an XmlDocument and writes out information describing each node (attribute and xml) in the tree.

  - See XmlDocExample in: xmlEgs.jar
3.d.5 Validating an Xml Document Using a Schema

- See the class `System.Xml.XmlValidatingReader`
  - Example program that validates an Xml Document against the corresponding Xml Schema
  - See `ValidatingXml` example in: `xmlEgs.jar`

- Input schema `bookStore.xsd` and `booksType.xsd`
  - `books.xml`
  - `booksOk.xml`
  - `booksFail.xml`

- Input schema `faculty.xsd`
  - `faculty.xml`
  - `facultyFail.xml`

- Input schema `message.xsd`
  - `message.xml`
3.e XML Stylesheet Language for Transformations (XSLT)

3.e.1 What is XSLT

- A stylesheet language used to process XML, with many potential uses:
  - Generate pdf from XML,
  - Generate postscript from XML,
  - Most common use is to generate HTML from XML.
  - See: [http://www.w3.org/1999/XSL/Transform](http://www.w3.org/1999/XSL/Transform)

- A specific transform in XSLT is an XML document, generally in a file with an xsl file extension, such as myTransform.xsl.

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <!-- specify the conversion here -->
</xsl:stylesheet>
```
### 3.e.2 XSLT Form is Declarative (Rule-Based)

- **Stylesheet concepts**
  - A stylesheet contains templates that determine the mappings to HTML at the Xml element level.
  - Each template contains a match attribute that selects the elements for which the template will be applied.
  - Within a template, any tags that do not start with xsl: are simply copied to the output.
  - The default result of matching an element takes any text within the element and copies it to the output. This is how strings:
    
    ```xml
    <quiz>Please define and analyze the significance of XML to Web Services.</quiz>
    ```
    
    end up in the html output.

- The transform is defined by the **rule** that matches the **root** of the xml:

  ```xml
  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  ```

  - This says: apply the templates that follow (in the xsl file) and surround their resulting html with **Html** and **Body** tags.
3.e.3 Converting xml to html with XSLT

• If the Xml had only a single element with a string, such as

```xml
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet href="myTransformer.xsl" type="text/xsl"?>
<simpleQuiz>
    Please define and analyze the significance of XML to Web Services.
</simpleQuiz>
```

• Then in addition to the root rult, the only remaining template may simply be:

```xml
<xsl:template match="simpleQuiz">
    <b><xsl:apply-templates/></b>
</xsl:template>
```

- This bolds whatever matches in the `simpleQuiz` element.

- The default action for a matched element is to return its text.
3.e.4 Converting xml to html with XSLT

- You can select attribute values.
  - Suppose element quiz has a name attribute that you want to appear as the title of the html page. You may use:
    
    ```xml
    <xsl:template match="quiz">
      <html><head><title><xsl:value-of select="@name"/></title>
      <style>
        h2 {color:navy; font-family:tahoma }
      </style></head><body>
      <xsl:apply-templates/>
    </body></html></xsl:template>
    ```

- Suppose you want to pass bold, italics and underscores in xml to html:
  
  ```xml
  <xsl:template match="b|i|u">
    <xsl:element name="{name()}">
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
  ```
3.e.5  XSLT: Using C# to Drive the Conversion

- Process involves an xml file to be converted, an xslt stylesheet defining the conversion and the html file that gets generated.

- Steps in performing an XSLT conversion with C#
  - Create a DOM for the xml to be converted using XmlDocument class
  - Create an XPathNavigator for the root of the DOM
  - Create an XslCompiledTransform object and load the XSL style sheet into the transform object
  - Create an XmlTextWriter object as the output stream for the generated html
  - See namespaces: System.Xml, System.Xml.XPath and System.Xml.Xsl

```csharp
XmlDocument doc = new XmlDocument();
XmlTextReader reader = new XmlTextReader(xmlFileName);
reader.WhitespaceHandling = WhitespaceHandling.None;
doc.Load(reader);
XmlElement root = doc.DocumentElement;
XPathNavigator nav = root.CreateNavigator();
XslTransform xslTransform = new XslTransform();
xslTransform.Load("file://"+Directory.GetCurrentDirectory()+"/"+xslFileName);
XmlTextWriter tw = new XmlTextWriter(htmlFileName, null);
xslTransform.Transform(nav, null, tw, null);
```
3.e.6  XSLT: Example Conversion

• Here is example using basic aspects of XSLT
  - simpleQuiz.xml
  - simpleQuiz.xsl
  - Note, your browser may convert the xml file into html. You may need to view page source to get the xml and xsl for this example.
  - You can convert to html with a C# program using the XslTransform class.
  - SeeTextXmlToHtml in the project: xsltExample.jar

• Here is a more complex quiz example:
  - egXsltQuiz.xml
  - egXsltQuiz.xsl

• To run the conversion
  - extract the jar file and change to the resulting Xslt directory; build
  - change to the docs directory and execute with mono:
  - mono ./bin/TestXmlToHtml.exe simpleQuiz.xml simpleQuiz.xsl tmp.html
  - load the tmp.html file with your browser.
3.e.7  XSLT: Converting Xml Documentation to Html

- C# compiler produces Xml from documentation comments.
  - csc /doc:myDocs.xml MyProgram.cs
  - Predefined documentation tags were defined in the Background section of the course notes.

- We can use XSLT to convert the xml into html.

- Two conversion examples can be found in: xsltExample.jar
  - TestXmlToHtml: Use an xslt stylesheet to convert xml to html.
  - GenHtmlDocs: Shows an example conversion from C# compiler produced xml to html, based on modifications to Schafer’s stylesheets.
    - This example generates static html pages from the compiler’s Xml output.

- Sourceforge also has a documentation project that converts xml documentation into Windows help files.
  - See: http://sourceforge.net/projects/ndoc/
3.f SOAP Messaging

3.f.1 What is SOAP

- Simple Object Access protocol (SOAP)
  - A way to access remote objects by sending XML messages over http
- SOAP over http includes both a request header and a request body
- Here is a SOAP http request header
  - POST /calendar-request HTTP/1.1
    Host: roo.east.asu.edu
    Content-Type: text/xml; charset="utf-8"
    Content-Length: 507
    SOAPAction: "http://roo.poly.asu.edu/Cst556/Examples/soap-example"
- The SOAP request body is an XML SOAP envelope of the form
  - <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <!-- SOAP description of requested service goes here -->
  </SOAP-ENV:Envelope>
3.f.2 SOAP Envelope

• The request body Xml above is a SOAP envelope XML element
• The SOAP envelope is in the XML namespace defined by the schema at:
• **Exercise**: follow the above hotlink and read the schema
  - Note, if you use Netscape, you will need to **View -> Page Source** to see the schema definition.
• The envelope declares the encoding for the body as
  - [http://schemas.xmlsoap.org/soap/encoding/](http://schemas.xmlsoap.org/soap/encoding/)
  - This is a method for structuring the request that is suggested within the SOAP specification itself, known as the SOAP serialization.
• SOAP specifies both a form for transporting messages and the form for message structure.
• Some other encoding styles (message structures) are:
  - Web Distributed Data Exchange (**WDDX**),
  - XML Remote Procedure Call (**XML-RPC**),
  - Resource Description Framework (**RDF**), or Custom XML
3.f.3 Example SOAP Request

• The entire request is:

```xml
POST /cal-request HTTP/1.0
Host: roo.east.asu.edu
Content-Type: text/plain; charset="utf-8"
Content-Length: xxx

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <c:AddAppointment xmlns:c="http://roo.poly.asu.edu/Cst556/cal">
      <c:when>
        <c:DateNTime>
          <day>8</day><month>10</month><year>2002</year>
          <start>4:15pm</start><stop>5:30pm</stop>
        </c:DateNTime>
      </c:when>
      <c:what>Cst556 Class Session on XML Stylesheet Language</c:what>
    </c:AddAppointment>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
3.f.4 Example Soap Response

- The response for this request could be:

HTTP/1.0 200 OK
Server: PythonSimpleHTTP/2.0
Date: Tue, 2 Oct 2002 11:23:03 GMT
Content-type: text/xml; charset="utf-8"
Content-length: 400

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <c:AddAppointmentResponse xmlns:c="http://roo.poly.asu.edu/Cst556/cal"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
3.g Assignment/Exercise

3.g.1 Notes for In-Class Discussion of Displaying Xml

- Protected members of DisplayCntrls.cs - for access in subclass
  - menu items for File and Xml menus
  - TreeView object must be manipulated by HandleUserInput.cs
  - RichTextBox object must be manipulated by HandleUserInput.cs
  - The ContextMenu (for right click menu) dynamically reconfigured (MenuItems.Add(...) and MenuItems.Clear())

- Adding a delegate for user actions in HandleUserInput.cs
  - My exitClick method is a delegate for handling exitMenuItem clicks:

    ```csharp
    this.exitMenuItem.Click += new System.EventHandler(this.exitClick);
    ```

- Inside the delegate, how do I know which item was clicked?

    ```csharp
    private void elementClick(object sender, System.EventArgs e) {
        string entity = ((MenuItem)sender).Text;
    }
    ```
3.g.2  Notes for In-Class Discussion of XML Assignment

- **OpenFileDialog, SaveFileDialog**
  - how to filter for xml (xsd) files

    openFileDialog1.Filter="xml files (*.xml)|*.xml|"+"All files (*.*)|*.*";
    openFileDialog1.FilterIndex = 1;

    - How to set the initial directory for search

3.g.3 Manipulating TreeView

- How are Nodes in the tree represented and manipulated?
  - The **Nodes** property of a **TreeView** object holds a group of **TreeNode** objects in a **TreeNodeCollection** object. These are the nodes at level 0 of the tree. See: [treeDemoWeb.jar](#)
  - In general, each **TreeNode** in the tree can contain its own **TreeNodeCollection**, defining the tree at subsequent levels.
  - The **TreeNode** class also has a **Nodes** property that can contain its own **TreeNodeCollection**.
  - So, the expression below accesses a node at level 3 of the tree, assuming that the nodes in the path have been defined.

```
tree.Nodes[0].Nodes[0].Nodes[1].Nodes[0].Text);
```

- A **TreeNodeCollection** object has methods to **Add**, **Remove** and **Clear**. It also has an **Item** property to access individual subtrees.

- The **Item** property of a **TreeNodeCollection** is accessed via an indexer.
  - So, **root.Nodes[i]** is a **TreeNode** object
  - And, **root.Nodes** is a **TreeNodeCollection** object, allowing **root.Nodes.Add(...)**, or **root.Nodes.Remove(...)**
3.g.4 Sample Code for TreeView Manipulation

- Consider the code for building a tree of nodes for country, city and state as shown in the Form below.
3.g.5  Code to Build a Sample TreeView Tree

- Consider the following points:
  - Build with: `ant -f sampleBuild.xml build`
  - Use `BeginUpdate` and `EndUpdate` to not visually update incrementally.
  - Consider using the wait cursor for extensive updates.
  - `TreeNode tn = tree.Nodes[i]` moves down one level in the tree.
  - Here is the code for the application, which also includes code to parse the full-path, traversing nodes that match. See the method `Button1_Click` in the example C# program.

```csharp
    tree.BeginUpdate();
    tree.Nodes.Clear();
    int i = tree.Nodes.Add(new TreeNode("<root>"));
    TreeNode tn = tree.Nodes[i];
    i = tn.Nodes.Add(new TreeNode("<UnitedStates>"));
    int coIndx = tn.Nodes[i].Nodes.Add(new TreeNode("<Colorado>"));
    int azIndx = tn.Nodes[i].Nodes.Add(new TreeNode("<Arizona>"));
    TreeNode azNode = tn.Nodes[i].Nodes[azIndx];
    int mesaInd = azNode.Nodes.Add(new TreeNode("<Mesa>"));
    i = tn.Nodes.Add(new TreeNode("<Canada>"));
    // add separator onto end to ensure as many separators as segments
    string mesaPath = azNode.Nodes[mesaInd].FullPath+tree.PathSeparator;
    Console.WriteLine("Path to Mesa is: {0}",mesaPath);
    tree.EndUpdate();
    tree.ExpandAll();
```