XML Classes

XML Sample File

```
<?xml version="1.0"?>
<SuperProProductList>
    <Product ID="1" Name="Chair">
        <Price>49.33</Price>
        <Available>True</Available>
        <Status>3</Status>
    </Product>
    <Product ID="2" Name="Car">
        <Price>43399.55</Price>
        <Available>True</Available>
        <Status>3</Status>
    </Product>
    <Product ID="3" Name="Fresh Fruit Basket">
        <Price>49.99</Price>
        <Available>False</Available>
        <Status>4</Status>
    </Product>
</SuperProProductList>
```

The XML Classes

- .NET support XML classes by System.XML namespace.
  - Reading and writing XML directly using XmlTextWriter and XmlTextReader.
  - Dealing with XML as a collection of in-memory objects using the XDocument class.
  - The Xml control to transform XML content to displayable HTML.

The XML TextWriter

Demo

- XMLTextWriter class methods like WriteStartDocument() and WriteEndDocument().
- WriteStartElement() and WriteEndElement()
- WrtiteAttributeString()
- WrtiteString() – used to insert text content inside the element.

The XML TextReader

A node is a designation that includes comments, whitespace, opening tags, closing tags, content, and even the XML declaration at the top of your file.

The XML TextReader

- Read() method used to read the nodes.
- Nodes having properties like NodeType and Name.

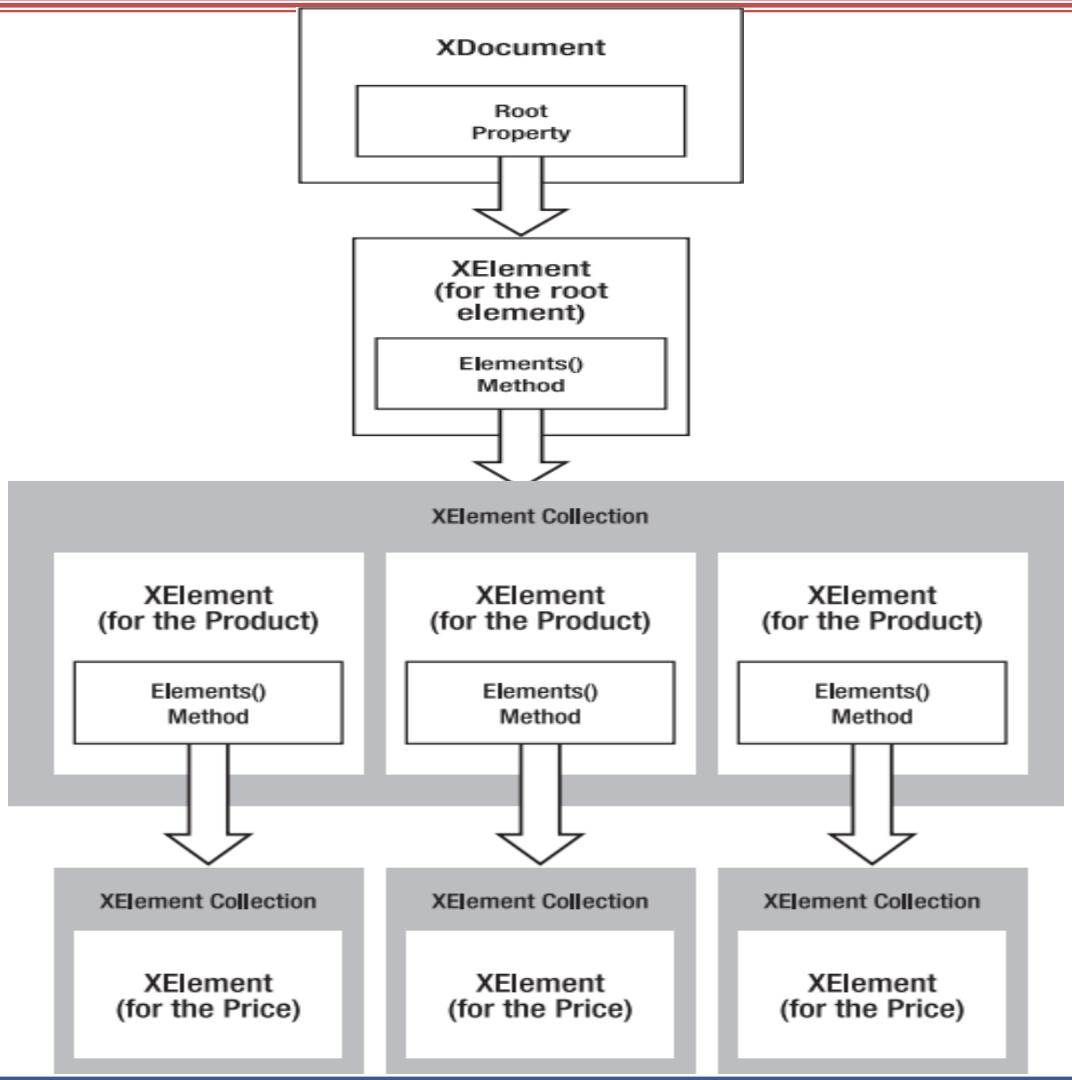
Demo

- The XmlTextReader and XmlTextWriter use XML as a backing store.

Working with XML Documents in Memory

- The XDocument class provides a different approach to XML data.
- It provides an in-memory model of an entire XML document.
- You can then browse through the entire document, reading, inserting, or removing nodes at any location.
- For these classes use System.XML.Linq namespace.

Working with XML Documents in Memory



The XML TextReader

- To start building a XML document, you need to create the XDocument, XElement, and XAttribute objects that comprise it.
- Example: XElement element = new XElement("Price", 23.99);

```
//Here's an example that creates an element with three nested elements and their content
XElement element = new XElement("Product",
                new XElement("ID", 3),
                new XElement("Name", "Fresh Fruit Basket"),
                new XElement("Price", 49.99)
                );
```

Here's the scrap of XML that this code creates:

```
<Product>
                <ID>3</ID>
                <Name>Fresh Fruit Basket</Name>
                <Price>49.99</Price>
</Product>
```

Demo

Reading an XML Document

- The XDocument makes it easy to read and navigate XML content.
- Xdocument.Load() method to read XML document.

Useful Methods for XElement and XDocument

Method	Description
Attributes()	Gets the collection of XAttribute objects for this element.
Attribute()	Gets the XAttribute with the specific name.
Elements()	Gets the collection of XElement objects that are contained by this element. (This is the top level only—these elements may in turn contain more elements.) Optionally, you can specify an element name, and only those elements will be retrieved.
Element()	Gets the single XElement contained by this element that has a specific name (or null if there's no match). If there is more than one matching element, this method gets just the first one.
Descendants()	Gets the collection of XElement objects that are contained by this element and (optionally) have the name you specify. Unlike the Elements() method, this method goes through all the layers of the document and finds elements at any level of the hierarchy.
Nodes()	Gets all the XNode objects contained by this element. This includes elements and other content, such as comments. However, unlike the XmlTextReader class, the XDocument does not consider attributes to be nodes.
DescendantNodes()	Gets all the XNode object contained by this element. This method is like Descendants() in that it drills down through all the layers of nested elements.

XElement class

```
// Load the document.
XDocument doc = XDocument.Load(file);
// Loop through all the nodes, and create the list of Product objects.
List<Product> products = new List<Product>();
foreach (XElement element in doc.Element("SuperProProductList").Elements("Product"))
{
        Product newProduct = new Product();
        newProduct.ID = (int)element.Attribute("ID");
        newProduct.Name = (string)element.Attribute("Name");
        newProduct.Price = (decimal) element.Element("Price");
        products.Add(newProduct);
}
// Display the results.
gridResults.DataSource = products;
gridResults.DataBind();
```

Searching an XML Document

- To search an XDocument, all you need to do is use the Descendants() or DescendantNodes() method.

Demo

XML Validation

- XML Schema defines the rules to which a specific XML document should conform, such as the allowable elements and attributes, the order of elements, and the data type of each element.
- You define these requirements in an XML Schema document (XSD).
- XML allows you to create a custom language for storing data, and XSD allows you to define the syntax of the language you create.

XML Namespaces

- Namespaces are disambiguate elements by making it clear what markup language they belong to.
- Example: you could tell the difference between your SuperProProductList standard and another organization's product catalog because the two XML languages would use different namespaces.

XML Namespaces

- Unique name to namespace like Universal Resource Identifiers (URIs).
- Example: http://www.mycompany.com/mystandard is a typical name for a namespace.
- To specify that an element belongs to a specific namespace, you simply need to add the xmlns attribute to the start tag and indicate the namespace.
- Example:

```
<Price xmlns="http://www.SuperProProducts.com/SuperProProductList">
49.33
</Price>
```

- When you assign a namespace in this fashion, it becomes the default namespace for all child elements.

```
<?xml version="1.0"?>
<SuperProProductList  xmlns="http://www.SuperProProducts.com/SuperProProductList">
   <Product>
      <ID>1</ID>
      <Name>Chair</Name>
      <Price>49.33</Price>
      <Available>True</Available>
      <Status>3</Status>
   </Product>
</SuperProProductList>
```

XML Namespaces

- To differentiate namespaces using namespace prefixes.
- Define the prefix in the xmlns attribute by inserting a colon (:) followed by the characters you want to use for the prefix.

```
<?xml version="1.0"?>
<super:SuperProProductList  xmlns:super="http://www.SuperProProducts.com/SuperProProductList">
   <super:Product>
      <super:ID>1</super:ID>
      <super:Name>Chair</super:Name>
      <super:Price>49.33</super:Price>
      <super:Available>True</super:Available>
      <super:Status>3</super:Status>
   </super:Product>

   <!-- Other products omitted. -->
</super:SuperProProductList>
```

Writing XML Content with Namespaces

- The XmlTextWriter includes an overloaded version of the WriteStartElement() method that accepts a namespace URI.

```
string ns = "http://www.SuperProProducts.com/SuperProProductList";
w.WriteStartDocument();
w.WriteStartElement("SuperProProductList", ns);
// Write the first product.
w.WriteStartElement("Product" , ns);
```

Writing XML Contents with Namespaces

- In XDocument Class, first create object of XNamespace.
- Then, you add this XNamespace object to the beginning of the element name every time you create an XElement (or an XAttribute) that you want to place in that namespace.

```
XNamespace ns = "http://www.SuperProProducts.com/SuperProProductList";

XDocument doc = new XDocument(new XDeclaration("1.0", null, "yes"),

new XComment("Created with the XDocument class."),
new XElement(ns + "SuperProProductList",
new XElement(ns + "Product",
new XAttribute("ID", 1),
new XAttribute("Name", "Chair"),
new XElement(ns + "Price", "49.33")
),
. . .
```

- In XMLTextReader class uses NamespaceURI property.
- If you're using the XDocument class, you need to take the XML namespace into account when you search the document.

XML Schema Definition

- All the XSD elements are placed in the http://www.w3.org/2001/XMLSchema namespace.
- This namespace uses the prefix xsd: or xs:

Demo

Validating an XML Document

- The first step when performing validation is to import the System.Xml.Schema namespace, which contains types such as XmlSchema and XmlSchemaCollection.
- **Two steps to create the validating reader:**
- Create an XmlReaderSettings object that specifically indicates you want to perform validation.

```
// Configure the reader to use validation.
XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;

// Create the path for the schema file.
string schemaFile = Path.Combine(Request.PhysicalApplicationPath, @"App_Data\SuperProProductList.xsd");

// Indicate that elements in the namespace http://www.SuperProProducts.com/SuperProProductList should be validated using
‘the schema file.
settings.Schemas.Add("http://www.SuperProProducts.com/SuperProProductList", schemaFile);
```

- Create the validating reader using the shared XmlReader.Create() method.

```
// Open the XML file.
FileStream fs = new FileStream(file, FileMode.Open);
// Create the validating reader.
XmlReader r = XmlReader.Create(fs, settings);
```

Demo

Validating an XML Document

Sample XSD File:

```
<?xml version="1.0"?>
<xs:schema
    targetNamespace="http://www.SuperProProducts.com/SuperProProductList"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" >
  <xs:element name="SuperProProductList">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="Product">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Price" type="xs:double" />
            </xs:sequence>
            <xs:attribute name="ID" use="required" type="xs:int" />
            <xs:attribute name="Name" use="required" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

END OF LECTURE