XML

CSIS 3540

Client Server Systems

Class 05

Topics

- XML Overview
- XML Document Model
- XML Support in .NET/C#
- Creating and reading XML files
- Reading and Writing objects to XML using Serialization

XML

- eXtensible Markup Language
- Provides a way to structure data
  - Use of a tree structure
- Data is surrounded by *tags*
  - Each tag has an *open* and a *close*
  - Data is placed in between the open/close tags
- Tags have names
- HTML is a tag-based language for describing web page data
  - XML is more general

XML Tags

- Have the format
  - <TagName> for the *open* tag
  - </TagName> for the *close* tag
- Data is placed between the open and close tag
- For the purpose of this course, only text data will be used
  - There are ways to specify data types for tags as well.
- Example
  - <Movie>Star Wars</Movie>
  - <Company>Microsoft</Company>
- Human readable and verbose

XML Document

- An XML document is a Tree with nested Elements
- Root
    - Element1
    - Element2
        - SubElementA
        - SubElementB
    - Element3
- Can be thought of as a database
- Easier to understand than a comma-separated-file
- Emerging standard for representing data

XML Document

- Consists of tag pairs, which can include other tag pairs.
- Forms a Tree structure

```
<Root>
   <Element>
      <SubElementA>Some data</SubElementA>
      <SubElementB>Some  data</SubElementB>
      …
      <SubElementZ>Some  data</SubElementZ>
   </Element>
   <Element>
      <SubElementA>Some data</SubElementA>
      <SubElementB>Some  data</SubElementB>
      …
      <SubElementZ>Some  data</SubElementZ>
   </Element>
<\Root>
```

Example: Houses

- See example below

```
<HouseListings>
          <House>
                    <HouseCode>NW01</HouseCode>
                    <HouseType>Condo</HouseType>
                    <Neighborhood>Uptown</Neighborhood>
                    <Price>200000</Price>
                    <MonthlyFees>125</MonthlyFees>
                    <Bedrooms>1</Bedrooms>
          </House>
          <House>
                    <HouseCode>BBY05</HouseCode>
                    <HouseType>House</HouseType>
                    <Neighborhood>South Slope</Neighborhood>
                    <Price>980000</Price>
                    <MonthlyFees>520</MonthlyFees>
                    <Bedrooms>4</Bedrooms>
          </House>
</HouseListings>
```

XML in C#/.NET

- An Element is defined by an XElement Class and object
- Elements can contain other elements
  - XElement el = new XElement();
  - … read XML file
  - price = el.Element(“House”).Element.(“Price”);
- Read an XML tree from a file

```
XElement root = XElement.Load(fileName);
foreach(var el in root)
  Do something – for example, copy the elements
  to another List
```

XML support in C#/.NET

```
 if (openFileDialogXML.ShowDialog() == DialogResult.OK)
     {
         XElement root = XElement.Load(openFileDialogXML.FileName);
// very simple

         foreach(var node in root.Elements())  //iterate through the
elements under root
                {
             string val1, val2;

             val1 = node.Element(“TagName1”).Value;
             val2 = node.Element(“TagName2”).Value;

             listBoxOutput.Items.Add(val1 + " " + val2);

          }
      }
```

Creating a simple XML Document

- Create a new XElement, which serves as Root node
- Add elements to the root
- Use Add() method to add more XElements
- Use Save(filename) method to write file
- See LinqToXMLFirstLook

```
 XElement inventoryDoc =
  new XElement("Inventory",
    new XComment("Current Inventory of cars!"),
        new XElement("Car", new XAttribute("ID", "1"),
        new XElement("Color", "Green"),
        new XElement("Make", "BMW"),
        new XElement("Name", "Stan")
      ),
      new XElement("Car", new XAttribute("ID", "2"),
        new XElement("Color", "Pink"),
        new XElement("Make", "Yugo"),
        new XElement("Name", "Melvin")
      )
  );

inventoryDoc.Add(
        new XElement("Car", new XAttribute("ID", "5"),
            new XElement("Color", "Red"),
            new XElement("Make", "Honda"),
            new XElement("Name", "Henry")
        )
     );

// Save to disk.
inventoryDoc.Save("SimpleInventory.xml");
```

Creating an XML Document

- Create a new XDocument, which has the full XML header
- Add Elements
- Has a Root property
- See ConstructingXMLDocs

```
XDocument inventoryDoc =
             new XDocument(
               new XComment("Current Inventory of cars!"),
               new XProcessingInstruction("xml-stylesheet",
                 "href='MyStyles.css' title='Compact' type='text/css'"),
               new XElement("Inventory",
                 new XElement("Car", new XAttribute("ID", "1"),
                   new XElement("Color", "Green"),
                   new XElement("Make", "BMW"),
                   new XElement("Name", "Stan")
                 ),
                 new XElement("Car", new XAttribute("ID", "2"),
                   new XElement("Color", "Pink"),
                   new XElement("Make", "Yugo"),
                   new XElement("Name", "Melvin")
                 ),
                   new XElement("Car", new XAttribute("ID", "5"),
                       new XElement("Color", "Red"),
                       new XElement("Make", "Honda"),
                       new XElement("Name", "Henry")
                   )
               )
             );

           // add another element
           inventoryDoc.Root.Add(
                   new XElement("Car", new XAttribute("ID", "7"),
                       new XElement("Color", "Orange"),
                       new XElement("Make", "Toyota"),
                       new XElement("Name", "Tomi")
                   )
                );
```

A complete App

- See LinqToXMLWinApp
- Reads an XML file using XDocument.Load(filename)
- Use LINQ
  - Insert new XElements
  - Query

Serialization

- Loads objects from and saves objects to XML directly and simply!
- Seems advanced, but is MUCH easier than the above.
- Declare XML usings, and then annotate classes using [Serializable]
- See XMLSerialize

```
using System.Xml;
using System.Xml.Serialization;
using System.IO;
 [Serializable]
    public class Car
    {
        public Car(string name, bool hatchBack)
        {
            Name = name;
            isHatchBack = hatchBack;
        }
        public Car() { }
        public string Name { get; set; }

        public Radio theRadio = new Radio();
        public bool isHatchBack;
    }
```

Writing an object to XML

- Create an XmlSerializer object of the type you want to write
- Create a StreamWriter stream
- Use the XmlSerializer.Serialize() method to convert the object to XML
- DONE!

```
static void SaveObjectAsXML(object myCar, Type carType, string fileName)
       {
           // Save object to a file named CarData.xml in XML format.

           XmlSerializer carSerializer = new XmlSerializer(carType);

           StreamWriter fileStream = new StreamWriter(fileName);

           carSerializer.Serialize(fileStream, myCar);
           fileStream.Close();
           Console.WriteLine($"=> Saved car in {fileName} in XML format");
       }
```

Reading an object from XML

- Create an XmlSerializer object of the type you want to read
- Create a StreamReader stream
- Use the XmlSerializer.Deserialize() method to convert XML to the object
    - Note need to cast
- DONE!

```
List<SecretAgentCar> myCars;

StreamReader carsFile = new StreamReader(fileName);

XmlSerializer xmlFormat = new XmlSerializer(typeof(List<SecretAgentCar>));

myCars = xmlFormat.Deserialize(carsFile) as List<SecretAgentCar>;
carsFile.Close();
```

XML Summary

- A “Leaf” tag has no children.
- A “Parent” tag is a single object
- HouseListings is a set of Houses, or a List
  - o XML Serializer will create ArrayOfHouses by default.
- House is a House object consisting of a set of leaf tags indicating properties of the object
  - o IE, Bedrooms
- Much easier to understand than CSV
- Used as a database