INTRODUCTION LINQ to XML

- The .NET Framework uses XML extensively.
- Configuration files use XML format.
- XML is also used heavily in serialization.
- **LINQ to XML** provides a way to manipulate data in XML documents using the same LINQ syntax you can use on arrays, collections, and databases.
- LINQ to XML also provides a set of classes for easily navigating and creating XML documents in your code.

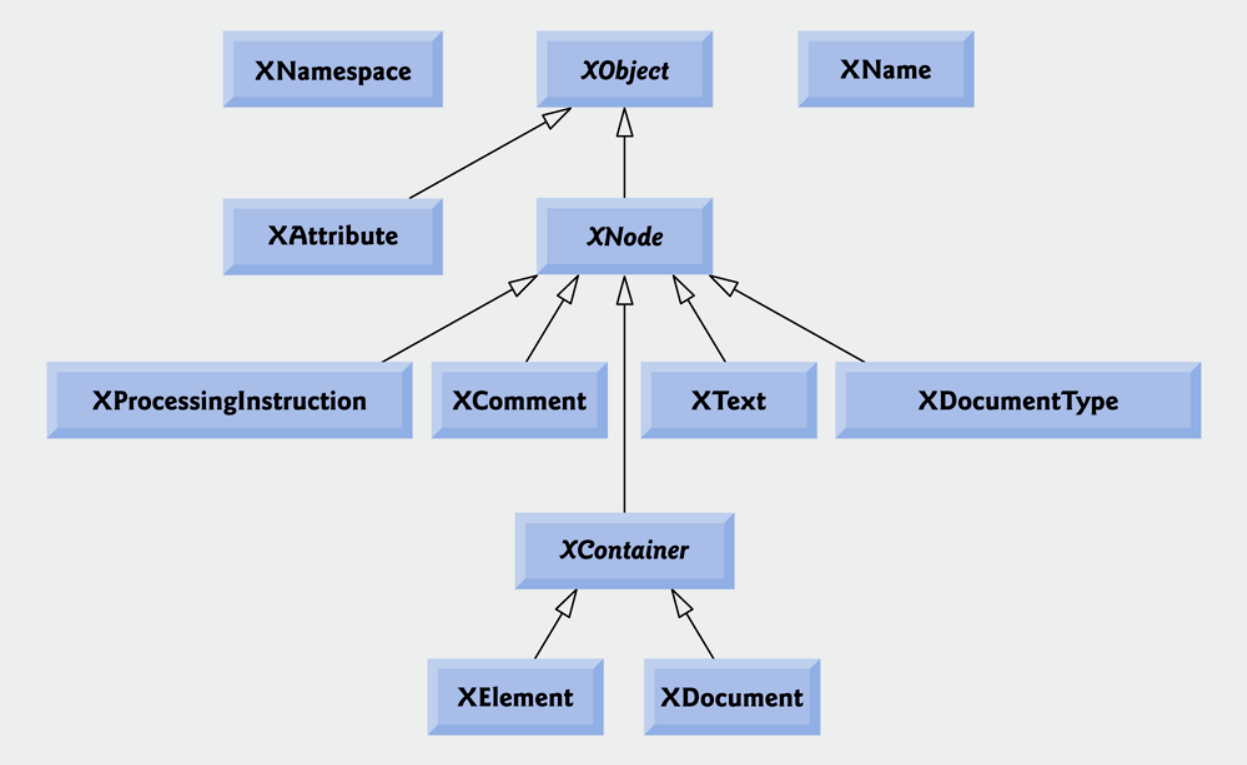
XML files DOM

- This hierarchical tree structure is called a Document Object Model (DOM) tree, and an XML parser that creates such a tree is known as a DOM parser.
- Each element name is represented by a node.
- A node that contains other nodes (called child nodes or children) is called a parent node.
- A parent node can have many children, but a child node can have only one parent node.
- Nodes that have the same parent are called sibling nodes.
- A node's descendant nodes include its children, its children's children and so on.
- A node's ancestor nodes include its parent, its parent's parent and so on.
- The DOM tree has a single root node, which contains all the other nodes in the document.
- Classes for creating, reading and manipulating XML documents are located in the System.Xml namespace.

Reading XML files

- Namespace **System.Xml.Linq** contains the classes used to manipulate a DOM in .NET, referred to collectively as LINQ to XML.
- The **XElement** class represents a DOM element node—an XML document is represented by a tree of XElement objects.
- The **XDocument** class represents an entire XML document.
- XDocument's static **Load method** takes an XML document's filename and returns an XDocument containing a tree representation of the XML file.
- The XDocument 's **Root property** returns an XElement representing the root element of the XML file.
- Because full element names consist of namespace prefix and name parts, tag and attribute names are stored as objects of class **XName**.
- The **Name property** of an XElement returns an XName object containing the tag name and namespace
    - The unqualified name is stored in the XName's **LocalName property**.
- The **HasElements property** can be used to test whether an elements has any child elements.
- The **Elements method** is used to obtain an element's children.
- An element's text can be obtained using the **Value property**.
- If used on an element with children, the Value property returns all of the text contained within its descendants, with the tags removed.

LINQ to XML Class Hierarchy



- The XDocument's Elements method can return all child elements, or only elements with a given tag name.
- The Elements method is actually defined in the **XContainer class**, the base class of XDocument and XElement.
- The **Descendants method** returns all descendant elements with the given tag name, not just direct children.
- The **Element method**, a member of the XContainer class, returns the first child element with the given tag name or null if no such element exists.
- The **Attribute method** of the XElement class returns an XAttribute object matching the given attribute name or null if no such object exists.
- The **XAttribute class** represents an XML attribute—it holds the attribute's name and value.

Before LINQ to XML we were used XMLDocument (XMLDOC) for manipulations in XML like adding attributes, elements and so on. **LINQ to XML uses XDocument** for the same kind of thing. Syntaxes are much easier than XMLDocument and it requires a minimal amount of code.

The mark up for Employees.xml is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<Employees>
 <Employee>
    <EmpId>1</EmpId>
    <Name>Sam</Name>
    <Sex>Male</Sex>
    <Phone Type="Home">423-555-0124</Phone>
    <Phone Type="Work">424-555-0545</Phone>
   <Address>
      <Street>7A Cox Street</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
 </Employee>
 <Employee>
    <EmpId>2</EmpId>
    <Name>Lucy</Name>
    <Sex>Female</Sex>
    <Phone Type="Home">143-555-0763</Phone>
    <Phone Type="Work">434-555-0567</Phone>
    <Address>
      <Street>Jess Bay</Street>
      <City>Alta</City>
      <State>CA</State>
      <Zip>95701</Zip>
      <Country>USA</Country>
    </Address>
 </Employee>
 <Employee>
    <EmpId>3</EmpId>
    <Name>Kate</Name>
    <Sex>Female</Sex>
    <Phone Type="Home">166-555-0231</Phone>
    <Phone Type="Work">233-555-0442</Phone>
    <Address>
      <Street>23 Boxen Street</Street>
      <City>Milford</City>
      <State>CA</State>
      <Zip>96121</Zip>
      <Country>USA</Country>
    </Address>
 </Employee>
 <Employee>
    <EmpId>4</EmpId>
    <Name>Chris</Name>
    <Sex>Male</Sex>
    <Phone Type="Home">564-555-0122</Phone>
```

```
    <Phone Type="Work">442-555-0154</Phone>
    <Address>
      <Street>124 Kutbay</Street>
      <City>Montara</City>
      <State>CA</State>
      <Zip>94037</Zip>
      <Country>USA</Country>
    </Address>
 </Employee>
</Employees>
```

Section 1: Read XML and Traverse the XML Document using LINQ To XML

1. How Do I Read XML using LINQ to XML

There are two ways to do so: Using the XElement class or the XDocument class. Both the classes contain the 'Load()' method which accepts a file, a URL or XMLReader and allows XML to be loaded. The primary difference between both the classes is that an XDocument can contain XML declaration, XML Document Type (DTD) and processing instructions. Moreover an XDocument contains one root XElement.

Using XElement

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
IEnumerable<XElement> employees = xelement.Elements();
// Read the entire XML
foreach (var employee in employees)
{
    Console.WriteLine(employee);
}
```

Output:

```
<Employee>
  <EmpId>1</EmpId>
  <Name>Sam</Name>
  <Sex>Male</Sex>
  <Phone Type="Home">423-555-0124</Phone>
  <Phone Type="Work">424-555-0545</Phone>
  <Address>
    <Street>7A Cox Street</Street>
    <City>Acampo</City>
    <State>CA</State>
    <Zip>95220</Zip>
    <Country>USA</Country>
  </Address>
</Employee>
<Employee>
  <EmpId>2</EmpId>
  <Name>Lucy</Name>
  <Sex>Female</Sex>
  <Phone Type="Home">143-555-0763</Phone>
```

```
  <Phone Type="Work">434-555-0567</Phone>
  <Address>
    <Street>Jess Bay</Street>
    <City>Alta</City>
    <State>CA</State>
    <Zip>95701</Zip>
    <Country>USA</Country>
  </Address>
</Employee>
<Employee>
  <EmpId>3</EmpId>
  <Name>Kate</Name>
  <Sex>Female</Sex>
  <Phone Type="Home">166-555-0231</Phone>
  <Phone Type="Work">233-555-0442</Phone>
  <Address>
    <Street>23 Boxen Street</Street>
    <City>Milford</City>
    <State>CA</State>
    <Zip>96121</Zip>
    <Country>USA</Country>
  </Address>
</Employee>

<Employee>
  <EmpId>4</EmpId>
  <Name>Chris</Name>
  <Sex>Male</Sex>
  <Phone Type="Home">564-555-0122</Phone>
  <Phone Type="Work">442-555-0154</Phone>
  <Address>
    <Street>124 Kutbay</Street>
    <City>Montara</City>
    <State>CA</State>
    <Zip>94037</Zip>
    <Country>USA</Country>
  </Address>
</Employee>
```

Using XDocument

```
XDocument xdocument = XDocument.Load("..\\..\\Employees.xml");
IEnumerable<XElement> employees = xdocument.Elements();
foreach (var employee in employees)
{
    Console.WriteLine(employee);
}
```

Output:

```
<Employees>
  <Employee>
    <EmpId>1</EmpId>
    <Name>Sam</Name>
    <Sex>Male</Sex>
    <Phone Type="Home">423-555-0124</Phone>
    <Phone Type="Work">424-555-0545</Phone>
    <Address>
      <Street>7A Cox Street</Street>
      <City>Acampo</City>
      <State>CA</State>
      <Zip>95220</Zip>
      <Country>USA</Country>
    </Address>
  </Employee>
  <Employee>
    <EmpId>2</EmpId>
    <Name>Lucy</Name>
```

Note 1: As you can observe, XDocument contains a single root element (Employee**s**).

Note 2: In order to generate an output similar to the one using XElement, use "xdocument.Root.Elements()" instead of "xdocument.Elements()"

Note 3: VB.NET users can use a new feature called XML Literal.

2. How Do I Access a Single Element using LINQ to XML

Let us see how to access the name of all the Employees and list them over here

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
IEnumerable<XElement> employees = xelement.Elements();
Console.WriteLine("List of all Employee Names :");
foreach (var employee in employees)
{
    Console.WriteLine(employee.Element("Name").Value);
}
```

Output:

```
List of all Employee Names :
Sam
Lucy
Kate
Chris
```

3. How Do I Access Multiple Elements using LINQ to XML

Let us see how to access the name of all Employees and also list the ID along with it

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
IEnumerable<XElement> employees = xelement.Elements();
Console.WriteLine("List of all Employee Names along with their ID:");
foreach (var employee in employees)
{
    Console.WriteLine("{0} has Employee ID {1}",
        employee.Element("Name").Value,
        employee.Element("EmpId").Value);
}
```

Output:

List of all Employee Names along with their ID:
Sam has Employee ID 1
Lucy has Employee ID 2
Kate has Employee ID 3
Chris has Employee ID 4

4. How Do I Access all Elements having a Specific Attribute using LINQ to XML

Let us see how to access details of all Female Employees

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
var name = from nm in xelement.Elements("Employee")
           where (string)nm.Element("Sex") == "Female"
           select nm;
Console.WriteLine("Details of Female Employees:");
foreach (XElement xEle in name)
    Console.WriteLine(xEle);
```

5. How Do I access Specific Element having a Specific Attribute using LINQ to XML

Let us see how to list all the Home Phone Nos.

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
var homePhone = from phoneno in xelement.Elements("Employee")
                where (string)phoneno.Element("Phone").Attribute("Type") ==
"Home"
                select phoneno;
Console.WriteLine("List HomePhone Nos.");
foreach (XElement xEle in homePhone)
{
    Console.WriteLine(xEle.Element("Phone").Value);
}
```

6. How Do I Find an Element within another Element using LINQ to XML

Let us see how to find the details of Employees living in 'Alta' City

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
var addresses = from address in xelement.Elements("Employee")
                where (string)address.Element("Address").Element("City") ==
"Alta"
               select address;
Console.WriteLine("Details of Employees living in Alta City");
foreach (XElement xEle in addresses)
    Console.WriteLine(xEle);
```

7. How Do I Find Nested Elements (using Descendants Axis) using LINQ to XML

Let us see how to list all the zip codes in the XML file

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
Console.WriteLine("List of all Zip Codes");
foreach (XElement xEle in xelement.Descendants("Zip"))
{
    Console.WriteLine((string)xEle);
}
```

Output:

8. How do I apply Sorting on Elements using LINQ to XML

Let us see how to List and Sort all Zip Codes in ascending order

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
IEnumerable<string> codes = from code in xelement.Elements("Employee")
                            let zip =
(string)code.Element("Address").Element("Zip")
                            orderby zip
                            select zip;
Console.WriteLine("List and Sort all Zip Codes");

foreach (string zp in codes)
    Console.WriteLine(zp);
```

Section 2: Manipulate XML content and Persist the changes using LINQ To XML

9. Create an XML Document with Xml Declaration/Namespace/Comments using LINQ to XML

When you need to create an XML document containing XML declaration, XML Document Type (DTD) and processing instructions, Comments, Namespaces, you should go in for the XDocument class.

```
XNamespace empNM = "urn:lst-emp:emp";

XDocument xDoc = new XDocument(
            new XDeclaration("1.0", "UTF-16", null),
            new XElement(empNM + "Employees",
                new XElement("Employee",
                    new XComment("Only 3 elements for demo purposes"),
                    new XElement("EmpId", "5"),
                    new XElement("Name", "Kimmy"),
                    new XElement("Sex", "Female")
                    )));

StringWriter sw = new StringWriter();
xDoc.Save(sw);
Console.WriteLine(sw);
```

```
<?xml version="1.0" encoding="utf-16"?>
<Employees xmlns="urn:lst-emp:emp">
  <Employee xmlns="">
    <!--Only 3 elements for demo purposes-->
    <EmpId>5</EmpId>
    <Name>Kimmy</Name>
    <Sex>Female</Sex>
  </Employee>
</Employees>
```

10. Save the XML Document to a XMLWriter or to the disk using LINQ to XML

Use the following code to save the XML to a XMLWriter or to your physical disk

```
XNamespace empNM = "urn:lst-emp:emp";

XDocument xDoc = new XDocument(
            new XDeclaration("1.0", "UTF-16", null),
            new XElement(empNM + "Employees",
                new XElement("Employee",
                    new XComment("Only 3 elements for demo purposes"),
                    new XElement("EmpId", "5"),
                    new XElement("Name", "Kimmy"),
                    new XElement("Sex", "Female")
                    )));

StringWriter sw = new StringWriter();
XmlWriter xWrite = XmlWriter.Create(sw);
```

```
xDoc.Save(xWrite);
xWrite.Close();

// Save to Disk
xDoc.Save("C:\\Something.xml");
Console.WriteLine("Saved");
```

11. Load an XML Document using XML Reader using LINQ to XML

Use the following code to load the XML Document into an XML Reader

```
XmlReader xRead = XmlReader.Create(@"..\\..\\Employees.xml");
XElement xEle = XElement.Load(xRead);
Console.WriteLine(xEle);
xRead.Close();

)
xRead.Close()
```

12. Find Element at a Specific Position using LINQ to XML

Find the 2nd Employee Element

```
// Using XElement
Console.WriteLine("Using XElement");
XElement xEle = XElement.Load("..\\..\\Employees.xml");
var emp1 = xEle.Descendants("Employee").ElementAt(1);
Console.WriteLine(emp);

Console.WriteLine("------------");

//// Using XDocument
Console.WriteLine("Using XDocument");
XDocument xDoc = XDocument.Load("..\\..\\Employees.xml");
var emp1 = xDoc.Descendants("Employee").ElementAt(1);
Console.WriteLine(emp);
```

13. List the First 2 Elements using LINQ to XML

List the details of the first 2 Employees

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
var emps = xEle.Descendants("Employee").Take(2);
foreach (var emp in emps)
    Console.WriteLine(emp);
```

14. List the 2nd and 3rd Element using LINQ to XML

List the 2nd and 3rd Employees

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
var emps = xEle.Descendants("Employee").Skip(1).Take(2);
foreach (var emp in emps)
    Console.WriteLine(emp);
```

15. List the Last 2 Elements using LINQ To XML

We have been posting the entire elements as output in our previous examples. Let us say that you want to display only the Employee Name, use this query:

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
var emps = xEle.Descendants("Employee").Reverse().Take(2);
foreach (var emp in emps)
    Console.WriteLine(emp.Element("EmpId") + "" + emp.Element("Name"));
```

To display only the values without the XML tags, use the 'Value' property

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
var emps = xEle.Descendants("Employee").Reverse().Take(2);
foreach (var emp in emps)
    Console.WriteLine(emp.Element("EmpId").Value + ". " +
emp.Element("Name").Value);
```

16. Find the Element Count based on a condition using LINQ to XML

Count the number of Employees living in the state CA

```
XElement xelement = XElement.Load("..\\..\\Employees.xml");
var stCnt = from address in xelement.Elements("Employee")
            where (string)address.Element("Address").Element("State") == "CA"
            select address;
Console.WriteLine("No of Employees living in CA State are {0}", stCnt.Count());
```

17. Add a new Element at runtime using LINQ to XML

You can add a new Element to an XML document at runtime by using the Add() method of XElement. The new Element gets added as the last element of the XML document.

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
xEle.Add(new XElement("Employee",
    new XElement("EmpId", 5),
    new XElement("Name", "George")));

Console.Write(xEle);
```

18. Add a new Element as the First Child using LINQ to XML

In the previous example, by default the new Element gets added to the end of the XML document. If you want to add the Element as the First Child, use the 'AddFirst()' method

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
    xEle.AddFirst(new XElement("Employee",
        new XElement("EmpId", 5),
        new XElement("Name", "George")));

    Console.Write(xEle);
```

19. Add an attribute to an Element using LINQ to XML

To add an attribute to an Element, use the following code:

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
    xEle.Add(new XElement("Employee",
        new XElement("EmpId", 5),
        new XElement("Phone", "423-555-4224", new XAttribute("Type",
"Home"))));

    Console.Write(xEle);
```

20. Replace Contents of an Element/Elements using LINQ to XML

Let us say that in the XML file, you want to change the Country from "USA" to "United States of America" for all the Elements. Here's how to do so:

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
    var countries =
xEle.Elements("Employee").Elements("Address").Elements("Country").ToList();
    foreach (XElement cEle in countries)
        cEle.ReplaceNodes("United States Of America");

    Console.Write(xEle);
```

21. Remove an attribute from all the Elements using LINQ to XML

Let us say if you want to remove the Type attribute (<Phone Type="Home">) attribute for all the elements, then here's how to do it.

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
    var phone = xEle.Elements("Employee").Elements("Phone").ToList();
    foreach (XElement pEle in phone)
        pEle.RemoveAttributes();

    Console.Write(xEle);


Console.Write(xEle)
```

To remove attribute of one Element based on a condition, traverse to that Element and SetAttributeValue("Type", null); You can also use SetAttributeValue(XName,object) to update an attribute value.

22. Delete an Element based on a condition using LINQ to XML

If you want to delete an entire element based on a condition, here's how to do it. We are deleting the entire Address Element

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
    var addr = xEle.Elements("Employee").ToList();
    foreach (XElement addEle in addr)
        addEle.SetElementValue("Address", null);

    Console.Write(xEle);
```

SetElementValue() can also be used to Update the content of an Element.

23. Remove 'n' number of Elements using LINQ to XML

If you have a requirement where you have to remove 'n' number of Elements; For E.g. To remove the last 2 Elements, then here's how to do it

```
XElement xEle = XElement.Load("..\\..\\Employees.xml");
    var emps = xEle.Descendants("Employee");
    emps.Reverse().Take(2).Remove();

    Console.Write(xEle);
```

24. Save/Persists Changes to the XML using LINQ to XML

All the manipulations we have done so far were in the memory and were not persisted in the XML file. You just need to call the Save() method. It's also worth observing that the structure of the code shown below is similar to the structure of the end result (XML document).

```
    XElement xEle = XElement.Load("..\\..\\Employees.xml");
    xEle.Add(new XElement("Employee",
    new XElement("EmpId", 5),
    new XElement("Name", "George"),
    new XElement("Sex", "Male"),
    new XElement("Phone", "423-555-4224", new XAttribute("Type", "Home")),
    new XElement("Phone", "424-555-0545", new XAttribute("Type", "Work")),
    new XElement("Address",
        new XElement("Street", "Fred Park, East Bay"),
        new XElement("City", "Acampo"),
        new XElement("State", "CA"),
        new XElement("Zip", "95220"),
        new XElement("Country", "USA"))));

    xEle.Save("..\\..\\Employees.xml");
    Console.WriteLine(xEle);

    Console.ReadLine();
```