Introduction to LINQ (3):

Language-Integrated Query

We have seen: LINQ to OBJECTS

and LINQ to DATA-TABLES

now:

LINQ to XML

XML (1)

- **XML** stands for E**X**tensible **M**arkup **L**anguage
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

Main characteristic:

Is a software- and hardware-independent tool for storing and transporting data. It simplifies data sharing, transport, and data sharing among different platforms.

HTML/XML Differences:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

All the major web browsers can read and display XML files.

XML (2)

```
<memo>
  <to>John</to>
  <from>Alexandra</from>
  <heading>To-Do List</heading>
  <body>Meeting at 9:30 at the College.</body>
</memo>
```

No Predefined Tags in XML

- The XML language has no predefined tags.
- The tags in the example above (like <to> and <from>) are not defined in any standard. -
- These tags are created by the author of the document.
- HTML works with predefined tags (like <p>, <h1>, <table>, etc).
- With XML, the author must define both the tags and the document structure.

Extensibility of XML

Most XML applications will work as expected even if new data is added or removed.
Imagine an application designed to display the original version of the “memo” (<to> <from> <heading> <data>).
Then imagine a newer version of with added <date> and <hour> elements, and a removed <heading> : everything will still work.

XML (3)

XML is widely used in web development.
In such setting, it is quite useful since it separates DATA from PRESENTATION.
This results in high re-usability of the code.
It complements the use of HTML.

Many informations exchange standards exist:

- Stocks and Shares
- Financial transactions
- Medical data
- Mathematical data
- Scientific measurements
- News information
- Weather services

The XML Tree Structure

XML documents are formed as **element trees**.
An XML tree starts at a **root** element and branches from the root to **child** elements.
All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The parent-child relation is used to describe the relationships between elements.
All elements can have text content.

XML Document Structure

XML uses a kind of self-describing syntax (remember: it is CASE SENSITIVE).
A **prolog** (optional) defines the XML version and the character encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The next line is the **root element** (which must be present) of the document:

```
<students>
```

The next line starts a <student> element (with an attribute value):

```
<student category="compsci">
```

The <students> elements have **4 child elements**:
<name>,< address>, <year>, <gpa>.

```
<name lang="en">John Smith</name>
<address>555 Maple Street</address>
<year>1983</year>
<gpa>99.99</gpa>
```

The next line ends the student element:

```
</student>
</students>
```

Some syntax

Pre-defined entity references in XML:

`<`	<	less than
`>`	>	greater than
`&`	&	ampersand
`&apos;`		apostrophe
`&quot;`	"	quotation mark

Comments:

```
<!-- This is a comment -->
```

White-space is Preserved in XML

XML does not truncate multiple white-spaces.
(HTML truncates multiple white-spaces to one single white-space).

The XML Element

The Element: `<tag> something </tag>`

The element can contain:
- text
- attributes
- other elements
- all the above

An element can be empty.
In that case, you can also use the self-closing element: `<element />`

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML Naming Rules

XML elements must follow these **naming rules**:

1) Element names are case-sensitive
2) Element names must start with a letter or underscore
3) Element names cannot start with the letters xml (or XML, or Xml, etc)
4) Element names can contain letters, digits, hyphens, underscores, and periods
5) Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Good Naming Practices

1) Create descriptive names, like this: `<person>, <firstname>, <lastname>`.
2) Create short and simple names, like this: `<book_title>` not like this: `<the_title_of_the_book>`.
3) Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".
4) Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".
5) Avoid ":". Colons are reserved for namespaces (more later).
6) Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them.

XML Attributes

XML elements can have attributes (they are optional), like HTML.
Attributes are designed to contain data related to a specific element.

XML attributes must be quoted.
Either single or double quotes can be used.
For a person's gender, the <person> element can be written like this:

```
<person gender="female">
```

or like this:

```
<person gender='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<person name='George "Gigi" Zeissler’>
```

or you can use character entities:

```
<person name="George &quot;Gigi&quot; Zeissler">
```

NOTE:

- attributes cannot contain multiple values (elements can)
- attributes cannot contain tree structures (elements can)
- attributes are not easily expandable (for future changes)

XML Prefixes and Namespaces (1)

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML code were added together, there would be a name conflict. Both contain a **<table>** element, but the elements have different content (and meaning).
Therefore, we introduce **prefixes**:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

XML Prefixes and Namespaces (2)

When using prefixes in XML, a **namespace** for the prefix must be defined.
The namespace can be defined by an **xmlns** attribute in the start tag of an element.
The namespace declaration has the following syntax. xmlns:prefix="URI".

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

XML Prefixes and Namespaces (3)

Namespaces can also be declared in the XML root element:

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

LINQ to XML

LINQ to XML

XML data can come from a variety of sources: web services, data files, configuration files.

LINQ to XML allows to:

- Create XML documents
- Modify XML documents (e.g. insert/delete/modify elements)
- Load/Save XML documents
- **query** XML documents.

Relevant namespaces:

`System.Xml.Linq` (—> we will use mainly this one)
`System.Diagnostics`
`System.Xml.Schema.Extensions`
`System.Xml.XPath.Extensions`

Most important classes:

`XDocument`
`XDeclaration`
`XElement`

Create a new XML document

```
XDocument xml_doc = new XDocument(
                new XDeclaration("1.0","utf-8","yes"),
                new XElement("Root","my_doc")
                );
```

The previous statement creates a new (well-formed) XML document containing no data.

For displaying the document (for example in a textbox):

```
txtBox.Text = xml_doc.Declaration.ToString()
              + " " +
              xml_doc.Document.ToString();
```

Namespaces

Namespaces provide an unique identification scheme for XML elements.
Namespaces are needed for avoiding conflicts among elements coming from different technologies/vendors/programmers.

```
//Define a namespace
XNamespace ns = “http://www.awebsite.ca”;

//Add it to the root node
xml_doc.Element(“Root”).Name = ns.GetName(“Root”);

//Add an element using the defined namespace
xml_doc.Element(ns + “Root”).Add(

                    new XElement(ns+”child1”,”some data”),
                    new XElement(ns+”child2”,”some data”),
                    ……
                )
```

Namespaces are accessed with the `XName` class.

Processing Instructions

Processing instructions tell the application processing the XML document, how exactly to process it.

```
//…create an XML file …
xml_doc.AddFirst(                    //processing instructions go first
      new XProcessingInstruction
      (“xml-stylesheet”,”type=‘text/xsl’ href=‘myXSL.XSL’”)
);
```

data string

target string.
An XSLT file in this case.
XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally **XSLT** does this by transforming each XML element into an (X)HTML element. With **XSLT** you can add/remove elements and attributes to or from the output **file**.

The browser opening you xml document will look first for the myXSL.XSL file for processing it.

Attributes

Attributes add information to an XML element.
Attributes can contain:

- a namespace for identify/interpret the element
- a datatype for understanding which information the element handles.
- sometimes they store some kind of useful data

```
xml_doc.Element("Root").Add(
          new XAttribute("my_attribute","attribute value")
);
```

Comments and Text

```
//after creating a document, add a comment:
xml_doc.AddFirst(new XComment("comment 1"));

//Another comment
xml_doc.Element("Root").Add(new XComment("comment 2"));

//Another comment
xml_doc.Add(new XComment("comment 3"));

//Add some text data
xml_doc.Element("Root").Add(new XText("some text.."));
```

XCData

CData in an XML document refers to a portion of it which does not contain XML-like data. For example, binary data can be included.

```
xml_doc.Element("Root").Add(
new XElement("CDataElement",new XCData("this is my cdata")));
```

It is a good idea to place the CData into a new element.
CData will be ignored by the XML parser.

Load and Save

The methods “Load” and “Save” perform the required operations:

EXAMPLE:

```
xml_doc.Save(”myDocument.xml”);

XDocument xml_doc = XDocument.Load(“C:\\PurchaseOrder.xml”);
```

Notice the double backslash

Starting to Query an XML Document: XNodeDocumentOrderComparer

```
//create some elements
xml_doc.Element("Root").Add(
    new XElement("e1","how"),
    new XElement("e2","are"),
    new XElement("e3","you?"));

//create the comparer object
XNodeDocumentOrderComparer comp = new XNodeDocumentOrderComparer();

//get the elements
XElement e1 = xml_doc.Element("Root").Element("e1");
XElement e2 = xml_doc.Element("Root").Element("e2");

//do the comparison
int Result = comp.Compare(e1,e2);

//print the results
if (Result==0) txtbox.Text = "e1 is like e2";
else if (Result < 0) txtbox.Text = "e1 comes before e2";
else txtbox.Text = "e2 comes before e1";
```

XNodeEqualityComparer

```
//create some elements
xml_doc.Element("Root").Add(
    new XElement("e1","how"),
    new XElement("e2","are"),
    new XElement("e3","you?"));

//create the comparer object
XNodeEqualityComparer comp = new XNodeEqualityComparer();

//get the elements
XElement e1 = xml_doc.Element("Root").Element("e1");
XElement e2 = xml_doc.Element("Root").Element("e2");

//do the comparison
int Result = comp.Equals(e1,e2);

//print the results
if (Result) txtbox.Text = "e1 equal e2";
else txtbox.Text = "e1 does not equal e2";
```

Remove

```
//create some elements
xml_doc.Element(“Root”).Add(
    new XElement(“e1”,”how”),
    new XElement(“e2”,”are”),
    new XElement(“e3”,”you?”));

//remove an element
xml_doc.Element(“Root”).Element(“e3”).Remove();
```

An XML document creation example

```
XDocument doc;

doc = new XDocument(
   new XDeclaration("1.0","utf-8","yes");

   new XElement("Contacts",

      new XElement("Contact",
         new XElement("Name","John Smith"),
         new XElement("Address","555 Maple Str"),
         new XElement("City","Vancouver")),

      new XElement("Contact",
         new XElement("Name","Jack Miller"),
         new XElement("Address","99 Str"),
         new XElement("City","New Westminster"))

   ) //close element "contacts"
); //close xdocument
```

Create a XML Document from Memory Objects

It is possible to create XML objects using collections already present in memory.
Using our usual “Students” example, let’s consider the following class:

```
class Student{
    public int ID {get; set;}
    public string Name {get; set;}
    public double GPA {get; set;}

    public static Student[] GetStudents(){
        Student st[] = new Student[3];

        st[0] = new Student {ID=1;Name=“John”;GPA=5};
        st[1] = new Student {ID=2;Name=“Jack”;GPA=4.5};
        st[2] = new Student {ID=3;Name=“Al”  ;GPA=3.3};
    }
}
```

For simplicity, the GetStudents returns all the students but also create them.

Create a XML Document from Memory Objects

Now we can construct an XML document containing the students using a LINQ query.

```
XDocument xdoc = new XDocument(
    new XDeclaration("1.0","utf-8","yes"),
    new XCommnet("Students List"),

    new XElement("Students",
        from s in Student.GetStudents()
        select new XElement("Student",new XAttribute("ID",s.ID),
                    new XElement("Name",s.Name),
                    new XElement("GPA",s.GPA)
                )
);

xdoc.Save(@"U:\mydir\myprojects\xmldocs\students.xml");
```