**LINQ**

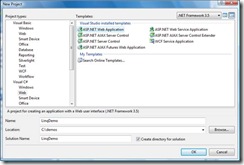
Here I’ll give you a quick introduction to LINQ using Visual Studio 2008. [Click here](http://msdn2.microsoft.com/en-us/netframework/aa904594.aspx) to learn more about LINQ in general, otherwise [download a 90-day trial Visual Studio 2008](http://msdn2.microsoft.com/en-us/vs2008/products/cc268305.aspx) (if you are a student you can download the professional edition for [FREE](https://downloads.channel8.msdn.com/)) and fire it up!

First we will look at some very basic LINQ to SQL query using code and the “Linq Relational Designer”, and then we’ll do another example to show you a code/query-free way of doing the same thing using the LinqDataSource.

**Linq to SQL (via code and Object Relational Designer)**

* 1. Open Visual Studio 2008 and create a new ASP.NET Web Application (here I’m using C#, but you could just as easily use VB.NET)

a. File-->New-->Project-->Visual C#/Web-->ASP.NET Web Application

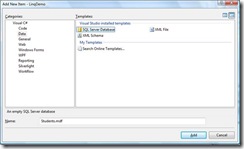
[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image002_2.jpg)

Once the project is loaded, we need to first add a database to the project (I am assume that you have [SQL Server 2005 Express Edition](http://www.microsoft.com/express/sql/Default.aspx) installed and running… it should have installed with Visual Studio 2008 by default, unless you selected otherwise).

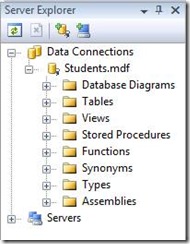
Now I’ll walk you through creating a sample database for this example or use your own existing database for the rest of this example.

2. Right-click the “App\_Data” folder and select Add-->New Item

3. Once the Add New Item windows opens select Data/SQL Server Database and name it Students.mdf as follows:

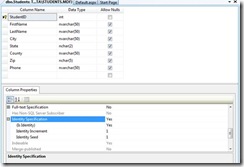
[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image004_2.jpg)

You should now be able to view you database in the “Server Explorer” window as it is below. Now we can start creating your database.

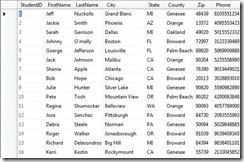
[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image006_2.jpg)

4. Right-click Tables and select Add New Table.

5. Define your columns as follows, be sure to make the StudentID column as an Identity field incrementing by 1.

[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image008_2.jpg)

6. Save your new table as “Students” and then populate it with some fictitious data…

[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image010_2.jpg)

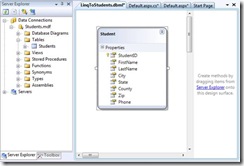
Now we can add a Linq to SQL Class…

7. From Solution Explorer, right-click you LinqDemo web application and select Add-->New Item-->Linq to SQL Classes

8. Name it LinqToStudents.dbml and click Add.

9. The Object Relational Designer should open.

10. From Server Explorer, drag and drop you Students table that you just created into the left pane of your designer surface. It should appear as follows:

[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image012_2.jpg)

11. Save All and Build for good measure…

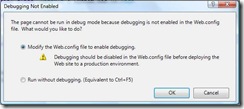
12. Back to the Default.aspx page let’s Drag and Drop a DataGrid server control from the Toolbox on to Designer View. (auto format it if you want, but don’t configure any datasources at this time).

13. Right-click anywhere on the Default.aspx page and select “View Code”, this should open the Default.aspx.cs class file.

14. Add the following code to the Page\_Load event.

|  |
| --- |
| LinqToStudentsDataContext linqStud = new LinqToStudentsDataContext();  var allstudents = from stud in linqStud.Students                   where stud.StudentID != null                   select stud;  GridView1.DataSource = allstudents; GridView1.DataBind(); |

15. Now you can click F5 (to run in Debug mode) or Ctrl + F5 (to run without the debugger), if you just click F5 for the first time you might get the following Information Windows letting you know that its modifying the Web.config file for you to allow debugging. Just click OK.

[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image014_2.jpg)

Your application should start up in Internet Explorer and display all of the data from your Students database table in the GridView control.

Albeit a basic example, you have just completed your first Linq To SQL application. CONGRATZ!

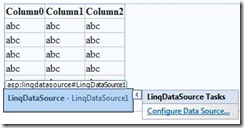
**Linq to SQL (via code and Object Relational Designer)**

Now that we did it the “hard” way (not really), let’s take a look at an even easier way to Linq our GridView control to a SQL table.

1. Using the same project and Default.aspx page in the previous example. Let’s first DELETE all the code you just added to the Page\_Load event of the Default.aspx.cs class file and then Save and Close that file.

2. From your Toolbox, drag and drop the LinqDataSource server control (under Data) anywhere onto the Designer View of your Default.aspx page.

3. You can now select “Configure Data Source” from the LinqDataSource you just added as follows:

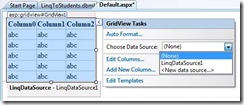
[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image016_2.jpg)

4. Once the “Configure Data Source” window appears you can the LinqDemo.LinqToStudentsDataContext option from the “Chose your context object” dropdown, and select Next.

5. On the next screen, just accept the defaults and click Finish.

Back to the Default.aspx Designer View… now we can bind the GridView to our newly created LinqDataSource.

6. Select “LinqDataSource1” from the Chose Data Source dropdown list of your GridView control as follows:

[](http://blogs.msdn.com/blogfiles/jeff/WindowsLiveWriter/StepbystepIntroductiontoLINQtoSQL_129E0/clip_image018_2.jpg)

LINQ or Language-Integrated Query is such a tool. LINQ is set of extensions to the .Net Framework 3.5 and its managed languages that sets the query as an object. It defines a common syntax and a programming model to query different types of data using a common language.

The relational operators like Select, Project, Join, Group, Partition, Set operations etc., are implemented in LINQ and the C# and VB compilers in the .Net framework 3.5, which support the LINQ syntax makes it possible to work with a configured data store without resorting to ADO.Net.

For example querying the Customers table in the Northwind database, using LINQ query in C#, the code would be:

|  |
| --- |
| var data = from c in dataContext.Customers  where c.Country == "Spain"  select c; |

Where:

* The 'from' keyword logically loops through the contents of the collection.
* The expression with the 'where' keyword is evaluated for each object in the collection.
* The 'select' statement selects the evaluated object to add to the list being returned.
* The 'var' keyword is for variable declaration. Since the exact type of the returned object is not known, it indicates that the information will be inferred dynamically.

LINQ query can be applied to any data-bearing class that inherits from IEnumerable<T>, here T is any data type, for example List<Book>.

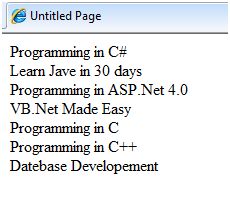
Let us look at an example to understand the concept. The example uses the following class: Books.cs

|  |
| --- |
| public class Books  {  public string ID {get; set;}  public string Title { get; set; }  public decimal Price { get; set; }  public DateTime DateOfRelease { get; set; }  public static List<Books> GetBooks()  {  List<Books> list = new List<Books>();  list.Add(new Books { ID = "001",  Title = "Programming in C#",  Price = 634.76m,  DateOfRelease = Convert.ToDateTime("2010-02-05") });    list.Add(new Books { ID = "002",  Title = "Learn Jave in 30 days",  Price = 250.76m,  DateOfRelease = Convert.ToDateTime("2011-08-15") });    list.Add(new Books { ID = "003",  Title = "Programming in ASP.Net 4.0",  Price = 700.00m,  DateOfRelease = Convert.ToDateTime("2011-02-05") });    list.Add(new Books { ID = "004",  Title = "VB.Net Made Easy",  Price = 500.99m,  DateOfRelease = Convert.ToDateTime("2011-12-31") });    list.Add(new Books { ID = "005",  Title = "Programming in C",  Price = 314.76m,  DateOfRelease = Convert.ToDateTime("2010-02-05") });    list.Add(new Books { ID = "006",  Title = "Programming in C++",  Price = 456.76m,  DateOfRelease = Convert.ToDateTime("2010-02-05") });    list.Add(new Books { ID = "007",  Title = "Datebase Developement",  Price = 1000.76m,  DateOfRelease = Convert.ToDateTime("2010-02-05") });  return list;  }  } |

The web page using this class has a simple label control, which will display the titles of the books. The Page\_Load event creates a list of books and returns the titles by using LINQ query:

|  |
| --- |
| public partial class simplequery : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  List<Books> books = Books.GetBooks();  var booktitles = from b in books select b.Title;  foreach (var title in booktitles)  lblbooks.Text += String.Format("{0} <br />", title);  }  } |

When the page is run, the label will display the results of the query:



The above LINQ expression:

|  |
| --- |
| var booktitles =  from b in books  select b.Title; |

Is equivalent to the following SQL query:

|  |
| --- |
| SELECT Title from Books |

# LINQ Operators:

Apart from the operators used so far, there are several other operators, which implement all query clauses. Let us look at some of the operators and clauses.

# The Join clause:

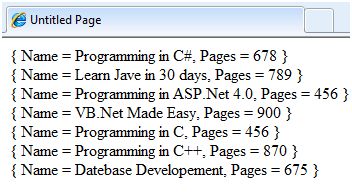
The 'join clause' in SQL is used for joining two data tables and displays a data set containing columns from both the tables. LINQ is also capable of that. To check this, add another class named Saledetails.cs in the previous project:

|  |
| --- |
| public class Salesdetails  {  public int sales { get; set; }  public int pages { get; set; }  public string ID {get; set;}  public static IEnumerable<Salesdetails> getsalesdetails()  {  Salesdetails[] sd =  {  new Salesdetails { ID = "001", pages=678, sales = 110000},  new Salesdetails { ID = "002", pages=789, sales = 60000},  new Salesdetails { ID = "003", pages=456, sales = 40000},  new Salesdetails { ID = "004", pages=900, sales = 80000},  new Salesdetails { ID = "005", pages=456, sales = 90000},  new Salesdetails { ID = "006", pages=870, sales = 50000},  new Salesdetails { ID = "007", pages=675, sales = 40000},  };  return sd.OfType<Salesdetails>();  }  } |

Add the codes in the Page\_Load event handler to query on both the tables using the join clause:

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  IEnumerable<Books> books = Books.GetBooks();  IEnumerable<Salesdetails> sales =  Salesdetails.getsalesdetails();  var booktitles = from b in books  join s in sales  on b.ID equals s.ID  select new { Name = b.Title, Pages = s.pages };  foreach (var title in booktitles)  lblbooks.Text += String.Format("{0} <br />", title);  } |

The resulted Page:

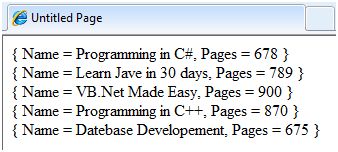


# The Where clause:

The 'where clause' allows adding some conditional filters to the query. For example, if you want to see the books, where the number of pages are more than 500, change the Page\_Load event handler to:

|  |
| --- |
| var booktitles = from b in books  join s in sales  on b.ID equals s.ID  where s.pages > 500  select new { Name = b.Title, Pages = s.pages }; |

The query returns only those rows, where the number of pages is more than 500:

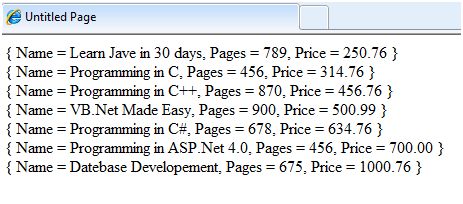


# The Orderby and Orderbydescending clauses:

These clauses allow sorting the query results. To query the titles, number of pages and price of the book, sorted by the price, write the following code in the Page\_Load event handler:

|  |
| --- |
| var booktitles = from b in books  join s in sales  on b.ID equals s.ID  orderby b.Price  select new { Name = b.Title,  Pages = s.pages, Price = b.Price}; |

The returned tuples are:



# The Let clause:

The let clause allows defining a variable and assigning it a value calculated from the data values. For example, to calculate the total sale from the above two sales, you need to calculate:

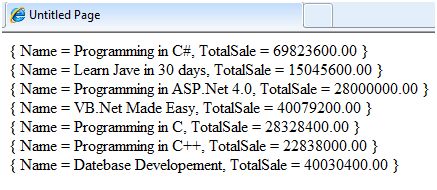
|  |
| --- |
| TotalSale = Price of the Book \* Sales |

To achieve this, add the following code snippets in the Page\_Load event handler:

The let clause allows defining a variable and assigning it a value calculated from the data values. For example, to calculate the total sale from the above two sales, you need to calculate:

|  |
| --- |
| var booktitles = from b in books  join s in sales  on b.ID equals s.ID  let totalprofit = (b.Price \* s.sales)  select new { Name = b.Title, TotalSale = totalprofit}; |

The resultant query page looks like:



**LINQ ON DATASET**

DataSet ds = new DataSet();

ds.Locale = CultureInfo.InvariantCulture;

FillDataSet(ds);

DataTable products = ds.Tables["Product"];

IEnumerable<DataRow> query =

from product in products.AsEnumerable()

select product;

Console.WriteLine("Product Names:");

foreach (DataRow p in query)

{

Console.WriteLine(p.Field<string>("Name"));

}

Fill the DataSet.

Dim ds As New DataSet()

ds.Locale = CultureInfo.InvariantCulture

' See the FillDataSet method in the Loading Data Into a DataSet topic.

FillDataSet(ds)

Dim contacts As DataTable = ds.Tables("Contact")

Dim query = \_

From contact In contacts.AsEnumerable() \_

Select contact \_

Order By contact.Field(Of String)("LastName")

Console.WriteLine("The sorted list of last names:")

For Each contact In query

Console.WriteLine(contact.Field(Of String)("LastName"))

Next

A lambda expression is an anonymous function that you can use to create delegates or expression tree types. By using lambda expressions, you can write local functions that can be passed as arguments or returned as the value of function calls. Lambda expressions are particularly helpful for writing LINQ query expressions.

To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator [=>](http://msdn.microsoft.com/en-us/library/vstudio/bb311046.aspx), and you put the expression or statement block on the other side. For example, the lambda expression x => x \* x specifies a parameter that’s named x and returns the value of x squared. You can assign this expression to a delegate type, as the following example shows:

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_b267d1b7-ab67-4978-9b8e-4d4a79c87cc3');" \o "Copy to clipboard.)

delegate int del(int i);

static void Main(string[] args)

{

del myDelegate = x => x \* x;

int j = myDelegate(5); //j = 25

}

To create an expression tree type:

C#

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_a13374cd-6b10-4d2a-bc8b-810c1fe624e1');)

using System.Linq.Expressions;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

Expression<del> myET = x => x \* x;

}

}

}

int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

int oddNumbers = numbers.Count(n => n % 2 == 1);

### Types of LINQ

Linq Comes with 3 Basic Types (Provided there are lots of more types of LINQ on different type of objects :

1. LINQ (Linq to Objects)2. DLINQ (Linq to SQL)3. XLINQ (Linq to XML)  
  
Before we continue with LINQ, we must look at some of the new inclusions which may come to us very handy.

var iNames = from i in employees

select i.name;

//Using Lambda Expression

var iNames = employees.Select<Employee, string>(r => r.name);

## What Is Lambda Expressions?

|  |  |
| --- | --- |
| 1 | (Input Parameters) => Method Expression |

don’t get mad just by having a glance at a lambda expression statement, It’s not an alien Language. It’s simply a **Function**(or delegate)! Well, we can call it as [anonymous function](http://msdn.microsoft.com/en-us/library/0yw3tz5k.aspx). It’s a function without a name. But it has parameter and return value, but they are dynamic, which means we can put anything for them, it’s on us. Wow! How flexible. Yes, it is.

Just remember one thing, as we have used it, its finish there, no way to reuse it. So, mostly, we will use this on some tasks which will need some computation and we want to write it easily but don’t need it again. If we need so, this must be assigned to a delegate or ‘Func’ type.

## General Usage Of C# Lambda Expression:

In general lambda expression is handy while using LinQ query in your application an apply them on a list of data, either from database, from xml or dynamic. In other words, we usual use it to filter out some specific data from a list of data that meets the criteria and use them on a LinQ operation.lambda expression can be used in wide range of area like linq to sql, linq to xml, linq to entity(Entity framework), array/list data etc

## A Simple Example:

Please take a look at the following expression:

|  |  |
| --- | --- |
| 1 | x => x >10 |

This is a very simple example how a lambda expression can be. The first part, before the ‘=>’ symbol, is refereed as the input parameter. It doesn’t need to be defined previously, its dynamic. It decides the type intelligently, even in compile time, when you are writing it, based on the operation being performed on it.

As we said, we want to filter out some specific data from a list of data, this input parameter stands for the type of each element of the list. Lets say, we are dealing with a list of integers data, then x is an integer. If we are dealing with a list of ‘A’ object, then x is also be of ‘A’ type and you will get full support of visual studio intelligent for it.

So lets see the complete statement of the above expression:

|  |  |
| --- | --- |
| 1  2 | int[] somevalues = { 10, 20, 5, 2, 40, 1 };  int numberOfExpectedValues = somevalues.Count(x => x >10); |

As you can see, the lambda expression is returning value which are being applied on ‘Count’ method of ‘somevalues’ array of integers. So, I mentioned, now x will be treated as an integer. And the expression will return a true/false value with testing each element of ‘somevalues’ array and will be used as delegate of count method. finally, in the ‘numberOfExpectedValues’ variable we will recieve how many data meet the criteria of being greater than 10.

## Another Pseudo Code Example:

How we can use it on a list of objects and how handy that would be? Lets see a pseudo code in c#(you have to write few more code to make it compile):

|  |  |
| --- | --- |
| 1  2  3  4 | List<User> myUsers = New List<User>();  //fill the list myUsers with some User object  List<User> activeUsers = myUsers.Where(u => u.Active == True);  //the activeUsers list will contain only active users filtered from list of all users |

So, without using any ugly SQL query or few lines of codes for a function to do this manually, we are having this in a single line of handy code. Isn’t it amazing?

**LINQ TO XML:**

For this article, we will be using a sample file called ‘Employees.xml’ for all our samples which is available with the [**source code**](http://cid-2c5f5b0560e374cb.office.live.com/self.aspx/.Public/Uploads/LINQ%20To%20XML%20Tutorial.zip). So make sure you keep it handy with you while are practicing these examples. 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>

The application is a console application targeting .NET 3.5 framework, although you can use the latest .NET 4.0 framework too. I have also used ‘query expressions’, instead of Lambda expression in these samples. It is just a matter of preference and you are free to use any of these.

This tutorial has been divided into 2 sections:

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

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

The following namespaces are needed while testing the samples: System; System.Collections.Generic; System.Linq; System.Text; System.Xml; System.Xml.Linq;

Go grab a hot cup of coffee, put on your developer cap and let us get started:

**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

C#

XElement xelement = XElement.Load("..\\..\\Employees.xml");

IEnumerable<XElement> employees = xelement.Elements();

// Read the entire XML

foreach (var employee in employees)

{

    Console.WriteLine(employee);

}

VB.NET (Converted Code)

Dim xelement As XElement = XElement.Load("..\..\Employees.xml")

Dim employees As IEnumerable(Of XElement) = xelement.Elements()

' Read the entire XML

For Each employee In employees

      Console.WriteLine(employee)

Next employee

Output:

Using XDocument

C#

XDocument xdocument = XDocument.Load("..\\..\\Employees.xml");

IEnumerable<XElement> employees = xdocument.Elements();

foreach (var employee in employees)

{

    Console.WriteLine(employee);

}

VB.NET (Converted Code)

Dim xdocument As XDocument = XDocument.Load("..\..\Employees.xml")

Dim employees As IEnumerable(Of XElement) = xdocument.Elements()

For Each employee In employees

      Console.WriteLine(employee)

Next employee

Output:

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

C#

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);

}

VB.NET (Converted Code)

Dim xelement As XElement = XElement.Load("..\..\Employees.xml")

Dim employees As IEnumerable(Of XElement) = xelement.Elements()

Console.WriteLine("List of all Employee Names :")

For Each employee In employees

      Console.WriteLine(employee.Element("Name").Value)

Next employee

Output:

**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

C#

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);

}

VB.NET (Converted Code)

Dim xelement As XElement = XElement.Load("..\..\Employees.xml")

Dim employees As IEnumerable(Of XElement) = xelement.Elements()

Console.WriteLine("List of all Employee Names along with their ID:")

For Each employee In employees

      Console.WriteLine("{0} has Employee ID {1}", employee.Element("Name").Value, employee.Element("EmpId").Value)

Next employee

Output:

**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

C#

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);

VB.NET (Converted Code)

Dim xelement As XElement = XElement.Load("..\..\Employees.xml")

Dim name = \_

      From nm In xelement.Elements("Employee") \_

      Where CStr(nm.Element("Sex")) = "Female" \_

      Select nm

Console.WriteLine("Details of Female Employees:")

For Each xEle As XElement In name

      Console.WriteLine(xEle)

Next xEle

Output:

**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.

C#

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);

}

VB.NET (Converted Code)

Dim xelement As XElement = XElement.Load("..\..\Employees.xml")

Dim homePhone = \_

      From phoneno In xelement.Elements("Employee") \_

      Where CStr(phoneno.Element("Phone").Attribute("Type")) = "Home" \_

      Select phoneno

Console.WriteLine("List HomePhone Nos.")

For Each xEle As XElement In homePhone

      Console.WriteLine(xEle.Element("Phone").Value)

Next xEle

Output:

**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

C#

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);

VB.NET (Converted Code)

Dim xelement As XElement = XElement.Load("..\..\Employees.xml")

Dim addresses = \_

      From address In xelement.Elements("Employee") \_

      Where CStr(address.Element("Address").Element("City")) = "Alta" \_

      Select address

Console.WriteLine("Details of Employees living in Alta City")

For Each xEle As XElement In addresses

      Console.WriteLine(xEle)

Next xEle

Output:

**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

C#

XElement xelement = XElement.Load("..\\..\\Employees.xml");

Console.WriteLine("List of all Zip Codes");

foreach (XElement xEle in xelement.Descendants("Zip"))

{

    Console.WriteLine((string)xEle);

}