ICT365

Software Development Frameworks

Dr Afaq Shah

Murdoch
UNIVERSITY

LINQ to XML

Murdoch
UNIVERSITY

Murdoch
UNIVERSITY

What is LINQ?

- Language Integrated Query
- Make query a part of the language
- Component of .NET Framework 3.5

Language-Integrated Query (LINQ)

Murdoch UNIVERSITY

- LINQ provides the ability to .NET languages (like C#, VB.NET, etc.) to generate queries to retrieve data from the data source.
- Traditionally, data is stored in a separate database from the application.
- Furthermore, you have to learn a different query language for each type of data source:
- SQL databases, XML documents, various Web services, and so on.
- With LINQ, more power given to the C# or .NET languages to generate a query for any LINQ compatible data source

Previously…

Murdoch UNIVERSITY

- We learned to connect to a database, set up a model of its tables and create queries like:

```
var x2 = db.Database.SqlQuery<Employee>("select * from employee");
```

- This is OK, but it has some disadvantages
  - We don't get support from the IDE for the query itself
  - It's an arbitrary string – no type checking
  - It's a bit of a pain to always need to figure out the type of the returned instances

The LINQ approach

Murdoch UNIVERSITY

- Makes the query actually a statement in the C# language

```
using (EmployeesContext db = new EmployeesContext())
          {
     var emps = from e in db.Employees
                          where e.DateOfBirth.Year < 1975
                          select e;
     string surname1=emps.First().surname
          }
```

We get access to all C#'s operators, too!

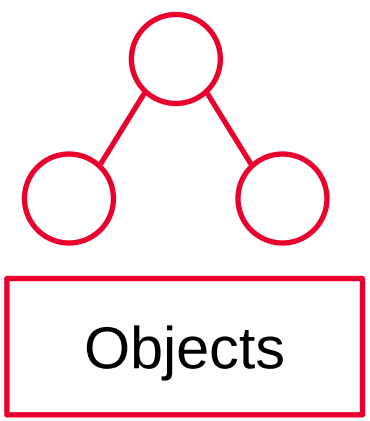
The LINQ Architecture

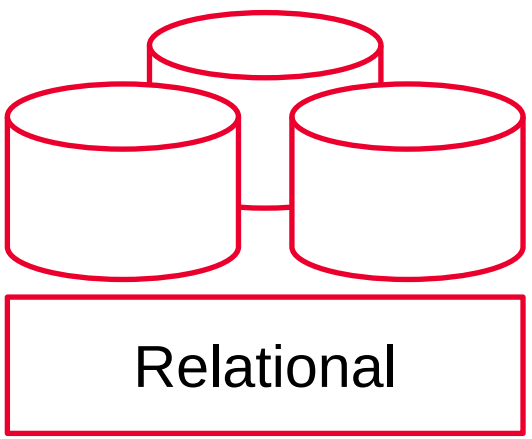
Murdoch UNIVERSITY

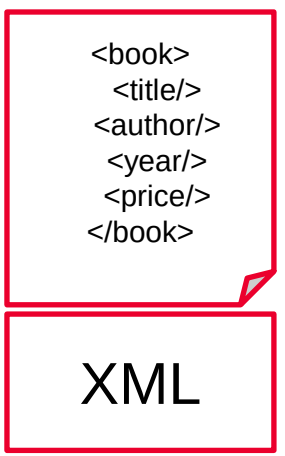
C# 3.0 | VB 9.0 | Others…

.NET Language Integrated Query

LINQ to Objects | LINQ to DataSets | LINQ to SQL | LINQ to Entities | LINQ to XML







Query expression

Murdoch UNIVERSITY

- For a developer who writes queries, the most visible "language-integrated" part of LINQ is the query expression.
- Query expressions are written in a declarative *query syntax*.
- By using query syntax, you can perform filtering, ordering, and grouping operations on data sources with a minimum of code.
- You use the same basic query expression patterns to query and transform data in SQL databases, ADO .NET Datasets, XML documents and streams, and .NET collections.

Query operation

```
class LINQQueryExpressions
{
        static void Main()
        {       // Specify the data source.
                int[] scores = new int[] { 97, 92, 81, 60 };
                // Define the query expression.
                IEnumerable<int> scoreQuery = from score in scores
                where score > 80 select score;
                // Execute the query.
                foreach (int i in scoreQuery)
                {  Console.Write(i + " ");  }
        }
}
// Output: 97 92 81
```

Enumerable class holds standard query operators that operate on object which executes *IEnumerable<T>*

Queries without LINQ

Murdoch UNIVERSITY

Objects using loops and conditions

```
foreach(Customer c in customers)
    if (c.Region == "USA") ...
```

SELECT from database tables

```
SELECT * FROM Customers WHERE
    Region='USA'
```

XML using XPath/XQuery

```
//Customers/Customer[@Region='USA'
```

Why LINQ?

Murdoch UNIVERSITY

- Traditional approaches - To find a specific object you need to write a large sum of code
- LINQ perform the same operation in a few numbers of lines
- Full type checking at compile time and helps us to detect the error at the runtime
- LINQ is simple, well-ordered, and high-level language than SQL
- You can also use LINQ with C# array and collections.
- With the help of LINQ you can easily work with any type of data source like XML, SQL, Entities, objects, etc.

Advantages of LINQ?

Murdoch UNIVERSITY

- User does not need to learn new query languages for a different type of data source or data format.
- It increase the readability of the code.
- Query can be reused.
- It gives type checking of the object at compile time.
- It can be used with array or collections.
- LINQ supports filtering, sorting, ordering, grouping.
- It makes easy debugging because it is integrated with C# language.
- It provides easy transformation means you can easily convert one data type into another data type like transforming SQL data into XML data.

LINQ Query

Murdoch UNIVERSITY

- What is a query?
- Different types of languages are developed to access different data sources e.g., SQL for a relational database, XQuery for XML
- using LINQ query you can access any type of data source like XML document, SQL database, ADO.NET dataset, etc

LINQ Query

```
using System;
using System.Linq;
class GFG {

static public void Main()
    {
        // Get data source
        string[] language = {"Cat", "Van", "Jam", "Cup",
                             "Car", "Pearl", "Rubber", "Piano"};

        // Create query
        var result = from lang in language
                     where lang.Contains('C')
                     select lang;

        // Execute Query
        foreach(var l in result)
        {
            Console.WriteLine(l);
        }
    }
}
```

from is used to specify the data source, where applies the filter and select provides the type of the returned items.

Output:
Cat
Cup
Car

LINQ Query

```
using System;
using System.Linq;
using System.Collections.Generic;

class GFG {

    static public void Main()
    {
        List<string> my_list = new List<string>() {
                "This is my Dog",
                "Name of my Dog is Robin",
                "This is my Cat",
                "Name of the cat is Mewmew"
        };

                var res = my_list.Where(a => a.Contains("Dog"));

        // Executing LINQ Query
        foreach(var q in res)
        {
            Console.WriteLine(q);
        }
    }
}
```

Output:
This is my Dog
Name of my Dog is Robin

LINQ to Objects

Murdoch UNIVERSITY

- Query any IEnumerable<T> source
  Includes arrays, List<T>, Dictionary...
- Many useful operators available
  Sum, Max, Min, Distinct, Intersect, Union
- Expose your own data with IEnumerable<T>
  or IQueryable<T>
- Create operators using extension methods

LINQ to Objects

Murdoch UNIVERSITY

- **LINQ to Objects** can be used to **filter** arrays and Lists, selecting elements that satisfy a set of conditions
- Repetition statements that filter arrays focus on the steps required to get the results. This is called **imperative programming**.
- LINQ queries, however, specify the conditions that selected elements must satisfy. This is known as **declarative programming**.
- The System.Linq namespace contains the LINQ to Objects provider.

Using LINQ with arrays

Murdoch UNIVERSITY

LINQWithSimpleTypeArray

```
// values greater than 4
var filtered = from value in values
               where value > 4
               select value;

var sorted = from value in values
             orderby value
             select value;

var sorted = from value in filtered
             orderby value descending
             select value;
```

A LINQ query begins with a **from clause**, which specifies a **range variable** (value) and the data source to query (values).

If the condition in the **where clause** evaluates to true, the element is selected.

The **select clause** determines what value appears in the results.

The **orderby clause** sorts the query results in ascending order.

The **descending** modifier in the orderby clause sorts the results in descending order.

Using LINQ with arrays of objects

- `Employee` class (Employee.cs)
- LINQWithArrayOfObjects

Let's fill the empty one ourselves together

An orderby clause can sort the results according to multiple properties, specified in a comma-separated list.

```
var nameSorted =
                    from e in employees
                    orderby e.LastName, e.FirstName
                    select e;
```

The query result's Any method returns true if there is at least one element, and false if there are no elements.

```
if (nameSorted.Any())
    Console.WriteLine(nameSorted.First().ToString() + "\n");
else
    Console.WriteLine("not found\n");
```

The query result's First method returns the first element in the result

Using LINQ with arrays of objects

Murdoch UNIVERSITY

```
Display( lastNames.Distinct(),
         "Unique employee last names");
```

The **Distinct method** removes duplicate elements, causing all elements in the result to be unique.

```
// use LINQ to select first and last names
var names =
            from e in employees
            select new { e.FirstName, Last = e.LastName };
```

The select clause can create a new object of **anonymous type** (a type with no name), which the compiler generates for you based on the properties listed in the curly braces ({}).

Using LINQ with `List<T>`

Murdoch UNIVERSITY

- **`List<T>`** is a generic list and its members are as follows:

Method / Property	Description
`Add`	Adds an object to the end of the List.
`Capacity`	Property that gets and sets the number of elements for which space is currently reserved in the List.
`Clear`	Removes all elements from the List.
`Contains`	Determines whether an element is in the List.
`Count`	Read-only property that gets the number of elements stored in the List.
`IndexOf`	Returns the zero-based index of the first occurrence of a value in the List
`Insert`	Inserts an element into the List at the specified index.
`Remove`	Removes the first occurrence of a specific object from the List.
`RemoveAt`	Removes the element at the specified index of the List.
`Sort`	Sorts the List.

Using LINQ with `List`'s

Murdoch UNIVERSITY

```
var startsWithR =
                    from item in items
                    let uppercaseString = item.ToUpper()
                    where uppercaseString.StartsWith("R")
                    orderby uppercaseString
                    select uppercaseString;
```

- LINQ's **let clause** can be used to create a new range variable to store a temporary result for use later in the LINQ query.

Example 1

Murdoch UNIVERSITY

```
using System;
using System.Linq;

class EXAMPLE {

    static public void Main()
    {
          int[] sequence = { 20, 30, 50, 78, 90, 79, 89, 99, 97, 29 };

        // Display the sequence
        Console.WriteLine("The Sequence is: ");

        foreach(int s in sequence)
        {
            Console.WriteLine(s);
        }

        // Finding sum of the given sequence
        // Using Sum function
        int result = sequence.Sum();
        Console.WriteLine("The sum of the given "
                              + "sequence is: {0}",
                          result);
    }
}
```

How will you find the sum of this sequence?

Example 2

Murdoch UNIVERSITY

```
using System;
using System.Linq;
using System.Collections.Generic;

// Employee details
public class Employee {

    public int emp_id
    {
        get;
        set;
    }

    public string emp_name
    {
        get;
        set;
    }

    public string emp_gender
    {
        get;
        set;
    }

    public string emp_hire_date
    {
        get;
        set;
    }

    public int emp_salary
    {
        get;
        set;
    }
}
```

Example 2

```
class EMP {

    // Main method
    static public void Main()
    {
        List<Employee> emp = new List<Employee>() {
            new Employee() { emp_id = 209, emp_name = "Anji", emp_gender = "Female", emp_hire_date =
    "12/3/2017", emp_salary = 20000 },
            new Employee() { emp_id = 210, emp_name = "Soni", emp_gender = "Female", emp_hire_date =
    "22/4/2018", emp_salary = 30000 },
            new Employee() { emp_id = 211, emp_name = "Robert", emp_gender = "Male", emp_hire_date =
    "3/5/2016", emp_salary = 40000 },
            new Employee() { emp_id = 212, emp_name = "Superwoman", emp_gender = "Female",
    emp_hire_date = "4/8/2017", emp_salary = 40000 },
            new Employee() { emp_id = 213, emp_name = "Rob", emp_gender = "Male", emp_hire_date =
    "12/1/2016", emp_salary = 40000 },
            new Employee() { emp_id = 214, emp_name = "Mary", emp_gender = "Female", emp_hire_date =
    "17/6/2015", emp_salary = 50000 },
        };

        // Count the total number of employees
        // Using Count () method
        var res = (from e in emp
                   select e.emp_id.Count();

        Console.WriteLine("Total number of Employees: {0}", res);
    }
}
```

LINQ to XML

Murdoch UNIVERSITY

- formerly known as XLinq
- Used for querying XML documents using LINQ syntax rather than the XPath/XQuery syntax.
- XML stands for eXtensible Markup Language
- markup language much like HTML
- designed to store and transport data
- designed to be self-descriptive

Using LINQ to XML, you can:

Murdoch UNIVERSITY

- Load XML from files or streams.
- Serialize XML to files or streams.
- Create XML trees from scratch using functional construction.
- Query XML trees using LINQ queries.
- Manipulate in-memory XML trees.
- Validate XML trees using XSD.
- Use a combination of these features to transform XML trees from one shape into another.

Murdoch
UNIVERSITY

System.Xml.Linq Namespace

- Contains the classes for LINQ to XML.

Classes

Extensions	Contains the LINQ to XML extension methods.
XAttribute	Represents an XML attribute.
XCData	Represents a text node that contains CDATA.
XComment	Represents an XML comment.
XContainer	Represents a node that can contain other nodes.
XDeclaration	Represents an XML declaration.
XDocument	Represents an XML document. For the components and usage of an XDocument object, see XDocument Class Overview.
XDocumentType	Represents an XML Document Type Definition (DTD).
XElement	Represents an XML element. See XElement Class Overview and the Remarks section on this page for usage information and examples.
XName	Represents a name of an XML element or attribute.
XNamespace	Represents an XML namespace. This class cannot be inherited.
XNode	Represents the abstract concept of a node (element, comment, document type, processing instruction, or text node) in the XML tree.
XNodeDocumentOrderComparer	Contains functionality to compare nodes for their document order. This class cannot be inherited.
XNodeEqualityComparer	Compares nodes to determine whether they are equal. This class cannot be inherited.
XObject	Represents a node or an attribute in an XML tree.
XObjectChangeEventArgs	Provides data for the Changing and Changed events.
XProcessingInstruction	Represents an XML processing instruction.
XStreamingElement	Represents elements in an XML tree that supports deferred streaming output.
XText	Represents a text node.

Enums

LoadOptions	Specifies load options when parsing XML.
ReaderOptions	Specifies whether to omit duplicate namespaces when loading an XDocument with an XmlReader.
SaveOptions	Specifies serialization options.
XObjectChange	Specifies the event type when an event is raised for an XObject.
LoadOptions	Specifies load options when parsing XML.
ReaderOptions	Specifies whether to omit duplicate namespaces when loading an XDocument with an XmlReader.

LINQ to XML contd…

Murdoch UNIVERSITY

- LINQ to XML provides an in-memory XML programming interface
- The LINQ family of technologies provides a consistent query experience for
  - objects (LINQ to Objects),
  - relational databases (LINQ to SQL), and
  - XML (LINQ to XML).

LINQ to XML contd…

Murdoch UNIVERSITY

- LINQ to XML is an up-to-date, redesigned approach to programming with XML.
- Provides the in-memory document modification capabilities
- Query expressions are syntactically different from Xpath but provide similar functionality.

LINQ to XML contd…

Murdoch UNIVERSITY

- LINQ to XML is like the Document Object Model (DOM).
- You can query and modify the document, and after you modify it you can save it to a file or serialize it and send it over the Internet.

LINQ family

Murdoch UNIVERSITY

- XML integration with Language-Integrated Query (LINQ).
- This integration enables you to write queries on the in-memory XML document
- The query capability of LINQ to XML is comparable in functionality (although not in syntax) to XPath and XQuery.
- LINQ provides stronger typing, compile-time checking, and improved debugger support

```xml
<?xml version="1.0"?>
<aw:PurchaseOrder
    aw:PurchaseOrderNumber="99503"
    aw:OrderDate="1999-10-20"
    xmlns:aw="http://www.adventure-works.com">
  <aw:Address aw:Type="Shipping">
    <aw:Name>Ellen Adams</aw:Name>
    <aw:Street>123 Maple Street</aw:Street>
    <aw:City>Mill Valley</aw:City>
    <aw:State>CA</aw:State>
    <aw:Zip>10999</aw:Zip>
    <aw:Country>USA</aw:Country>
  </aw:Address>
  <aw:Address aw:Type="Billing">
    <aw:Name>Tai Yee</aw:Name>
    <aw:Street>8 Oak Avenue</aw:Street>
    <aw:City>Old Town</aw:City>
    <aw:State>PA</aw:State>
    <aw:Zip>95819</aw:Zip>
    <aw:Country>USA</aw:Country>
  </aw:Address>
  <aw:DeliveryNotes>Please leave packages in shed by driveway.</aw:DeliveryNotes>
  <aw:Items>
    <aw:Item aw:PartNumber="872-AA">
      <aw:ProductName>Lawnmower</aw:ProductName>
      <aw:Quantity>1</aw:Quantity>
      <aw:USPrice>148.95</aw:USPrice>
      <aw:Comment>Confirm this is electric</aw:Comment>
    </aw:Item>
    <aw:Item aw:PartNumber="926-AA">
      <aw:ProductName>Baby Monitor</aw:ProductName>
      <aw:Quantity>2</aw:Quantity>
      <aw:USPrice>39.98</aw:USPrice>
      <aw:ShipDate>1999-05-21</aw:ShipDate>
    </aw:Item>
  </aw:Items>
</aw:PurchaseOrder>
```

Murdoch UNIVERSITY

Sample XML File:
Typical Purchase Order
in a Namespace

Murdoch UNIVERSITY

By using LINQ to XML, you could run the following query to obtain the part number attribute value for every item element in the purchase order:

```
IEnumerable<string> partNos =
        from item in purchaseOrder.Descendants("Item")
        select (string) item.Attribute("PartNumber");
```

Murdoch UNIVERSITY

…you might want a list, sorted by part number, of the items with a value greater than $100:

```
IEnumerable<XElement> partNos =

from item in purchaseOrder.Descendants("Item")

where (int) item.Element("Quantity") * (decimal)
item.Element("USPrice") > 100

orderby (string)item.Element("PartNumber")

select item;
```

Loading Xml Content

Murdoch UNIVERSITY

- Loading Xml is performed with;

XElement.Load

XDocument.Load

- Both support loading from

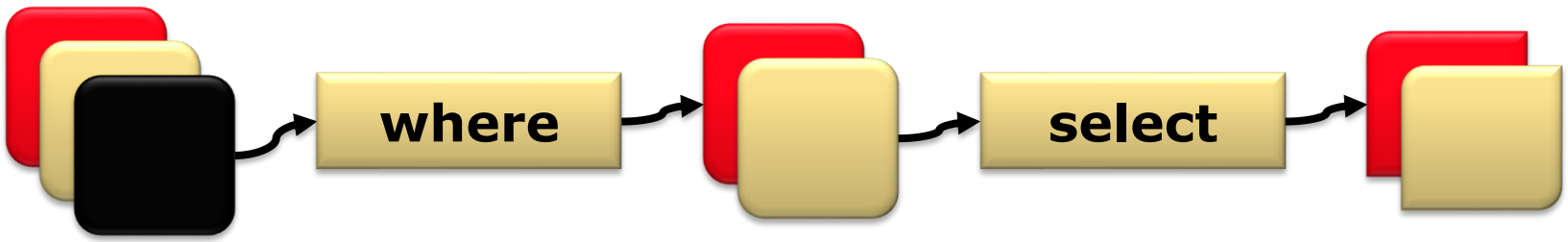
URI, XmlReader, TextReader

```
XmlReader reader = XmlReader.Create("myData.xml");

XElement element = XElement.Load(reader);
```

IEnumerable<T> & IQueryable<T>

Murdoch UNIVERSITY

- IEnumerable – query executed piece by piece



- IQueryable – query executed in one go

Modifying XML

Murdoch UNIVERSITY

- XML tree exposed by **XElement** and friends is not read-only
- Modifications through methods such as;

XElement.Add(), XElement.Remove(), etc.

- Modified tree can be persisted via

XElement.Save(), XDocument.Save()

Both supporting filename, TextWriter, XmlWriter.

```
XElement element = new XElement("foo");

element.Save(@"c:\temp\foo.xml");
```

Casting elements

- LINQ to XML code still contains quite a lot of **casts** and **strings**

```
var query = from x in element.Descendants("customer")
            where (string)x.Attribute("country") == "UK"
            select (int)x.Attribute("age");
```

- LINQ to XSD

Generates strongly typed classes from XSD

Derived from XElement, XDocument, etc.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/how-to-retrieve-the-value-of-an-element-linq-to-xml

LINQ to XML Example

XML construction

```
namespace LINQ_Learning
{
  public class Customers
   {
      public int CustID { get; set; }
      public string Name { get; set; }
      public long MobileNo { get; set; }
      public string Location { get; set; }
      public string Address { get; set; }

       public static List<Customers> GetCostomersDetail()
     {
        List<Customers> lstcustomer = new List<Customers>()
        {
           new Customers{CustID=10001,Name="Robert" , MobileNo=9820098200, Location="Balaga",Address="XYZ"},
            new Customers{CustID=10001,Name="Richard" , MobileNo=9820011234, Location="Nedlands",Address="ABC
"},
             new Customers{CustID=10001,Name="Kate" , MobileNo=9820011266, Location="Murdoch",Address="CDE"},

              new Customers{CustID=10001,Name="Catherine" , MobileNo=890012452, Location="Wembley",Address="M
NO"}
        };
          return lstcustomer;
     }
   }
}
```

LINQ to XML Example


```
namespace LINQ_Learning
{
    class Program
    {
        static void Main(string[] args)
        {
            XDocument xmlDocument = new XDocument(
                new XDeclaration("1.0","utf-8","yes"),

                new XComment("LINQ To XML Demo"),

                new XElement("Customers"),

from customers in Customers.GetCostomersDetail()
            select new XElement("Customer" , new XAttribute("ID" , customers.CustID),
                    new XElement("Name" , customers.Name),
                    new XElement("Mobile" , customers.MobileNo),
                    new XElement("Location" , customers.Location),
                    new XElement("Address" , customers.Address))

                ));
            xmlDocument.Save(@"C:\Dev\LINQ_Learning\CustomersDetail.xml");

            Console.ReadLine();
        }
    }
}
```

LINQ to XML Example

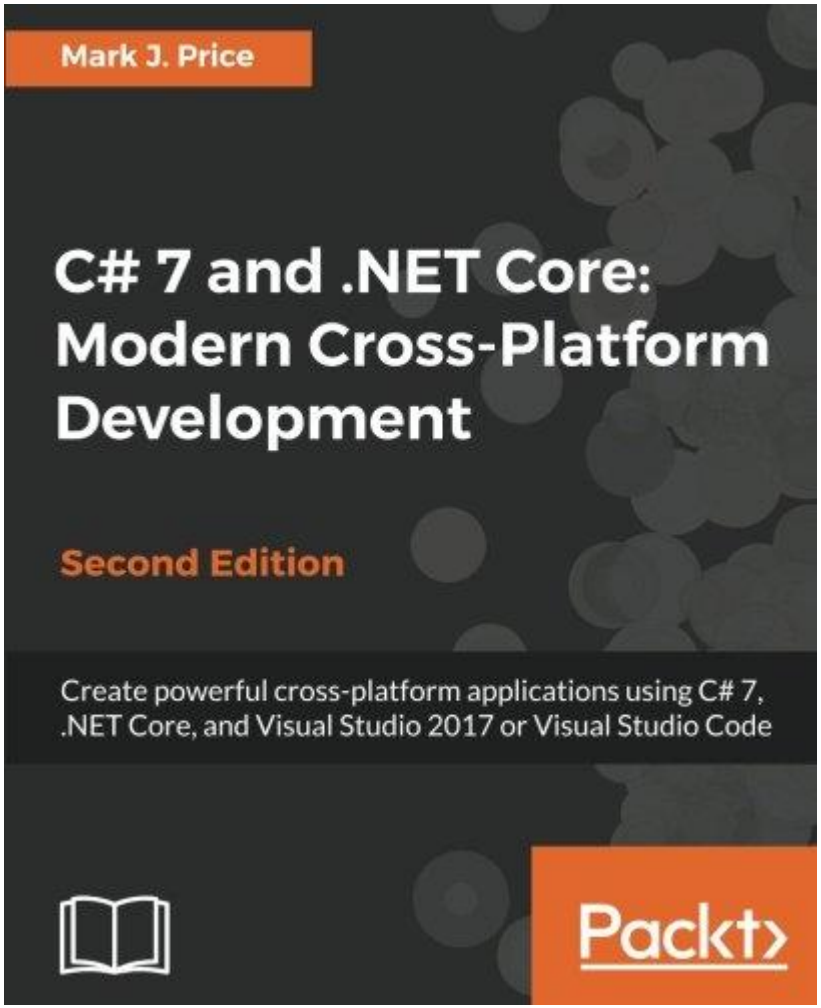

```
namespace LINQ_Learning
{
   class Program
   {
      static void Main(string[] args)
      {

        xmlDocument.Save(@"C:\Dev\LINQ_Learning\CustomersDetail.xml");

 IEnumerable<string> names = from customers in
                    XDocument.Load(@"C:\Dev\LINQ_Learning\CustomersDetail.xml")
                          .Descendants("Customer")
                   select customers.Element("Name").Value;

         foreach(string strName in names)
         {
            Console.WriteLine(strName);
         }

         Console.ReadLine();
      }
   }
}

}
```

Querying using LINQ to XML

Reading/ reference

Murdoch UNIVERSITY

Chapter 9. Querying and Manipulating Data with LINQ

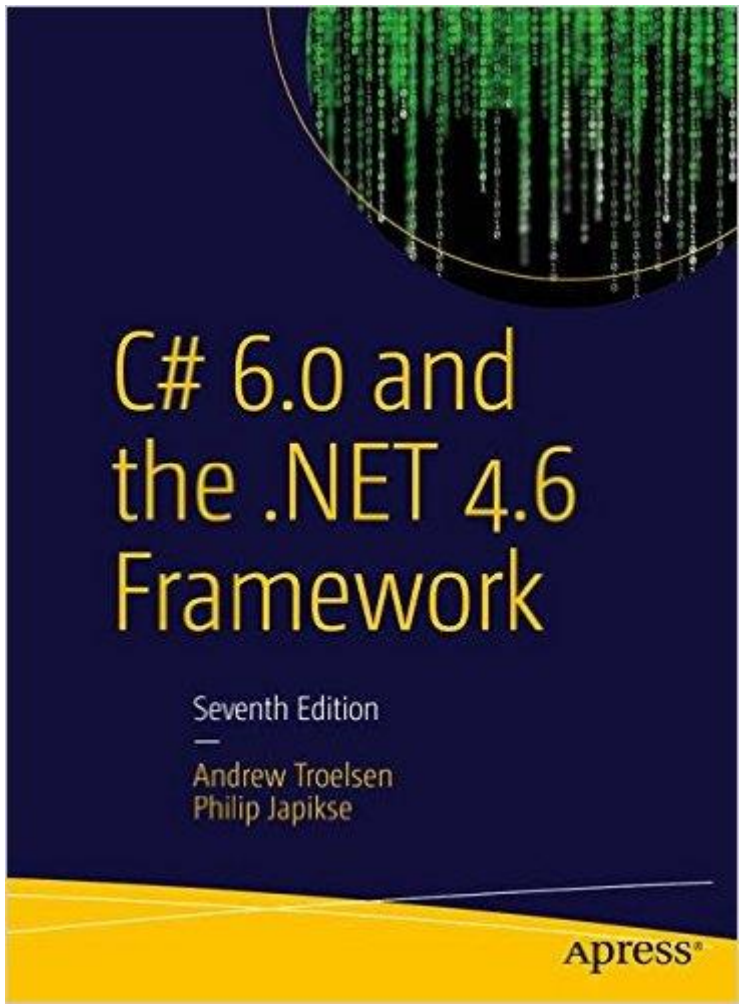


Reading/ reference

Murdoch UNIVERSITY

Chapter: LINQ to Objects

Chapter: Introducing LINQ to XML



For more information….

Murdoch UNIVERSITY

- MSDN Official site

  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/index

- MSDN 101 LINQ Samples

  http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx

- LINQ Pad

  http://www.linqpad.net/

Next steps

Murdoch UNIVERSITY

To learn more details about LINQ, start by becoming familiar with some basic concepts in Query expression basics:

https://docs.microsoft.com/en-us/dotnet/csharp/linq/query-expression-basics

Murdoch UNIVERSITY

Then read the documentation for the LINQ technology in which you are interested:

- XML documents: LINQ to XML:

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-xml

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/programming-guide-linq-to-xml

Murdoch UNIVERSITY

- ADO.NET Entity Framework: LINQ to entities:

https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/language-reference/linq-to-entities

- .NET collections, files, strings and so on: LINQ to objects:

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-objects

Murdoch UNIVERSITY

- To gain a deeper understanding of LINQ in general, see LINQ in C#:

https://docs.microsoft.com/en-us/dotnet/csharp/linq/linq-in-csharp

- To start working with LINQ in C#, see the tutorial Working with LINQ:

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/working-with-linq

Acknowledgement

Murdoch UNIVERSITY

- Sources used in this presentation include:
  - Geeksforgeeks
  - C-sharpcorner
  - MSDN