LinQ

language integrated query

Microsoft

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

The acronym LINQ stands for Language Integrated Query. Microsoft's query language is fully integrated and offers easy data access from in-memory objects, databases, XML documents, and many more. It is through a set of extensions LINQ ably integrates queries in C# and Visual Basic.

This tutorial offers a complete insight into LINQ with ample examples and coding. The entire tutorial is divided into various topics with subtopics that a beginner can be able to move gradually to more complex topics of LINQ.

Audience

The aim of this tutorial is to offer an easy understanding of LINQ to the beginners who are keen to learn the same.

Prerequisites

It is essential before proceeding to have some basic knowledge of C# and Visual Basic in .NET.

Copyright & Disclaimer

Table of Contents

1. LINQ – Overview

Developers across the world have always encountered problems in querying data because of the lack of a defined path and need to master a multiple of technologies like SQL, Web Services, XQuery, etc.

Introduced in Visual Studio 2008 and designed by Anders Hejlsberg, LINQ (Language Integrated Query) allows writing queries even without the knowledge of query languages like SQL, XML etc. LINQ queries can be written for diverse data types.

Example of a LINQ query

C#

```
using System;
using System.Linq;

class Program
{
   static void Main()
   {
      string[] words = {"hello", "wonderful", "LINQ", "beautiful", "world"};

      //Get only short words
      var shortWords = from word in words where word.Length <= 5 select word;

      //Print each word out

      foreach (var word in shortWords)
      {
         Console.WriteLine(word);
      }

      Console.ReadLine();
   }
}
```

VB

```
Module Module1

   Sub Main()
      Dim words As String() = {"hello", "wonderful", "LINQ", "beautiful", "world"}

      ' Get only short words
      Dim shortWords = From word In words _ Where word.Length <= 5 _ Select word

      ' Print each word out.

      For Each word In shortWords
         Console.WriteLine(word)
      Next

      Console.ReadLine()
   End Sub
End Module
```

When the above code of C# or VB is compiled and executed, it produces the following result:

```
hello
LINQ
world
```

Syntax of LINQ

There are two syntaxes of LINQ. These are the following ones.

Lamda (Method) Syntax

```
var longWords = words.Where( w => w.length > 10);
Dim longWords = words.Where(Function(w) w.length > 10)
```

Query (Comprehension) Syntax

```
var longwords = from w in words where w.length > 10;
Dim longwords = from w in words where w.length > 10
```

Types of LINQ

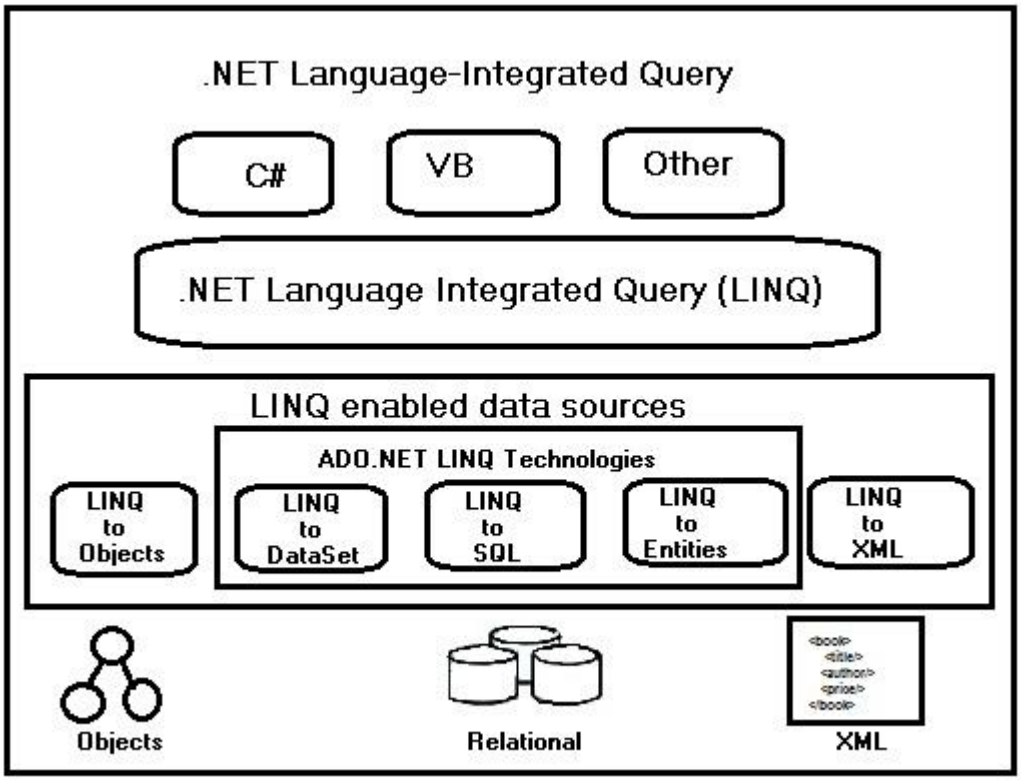
The types of LINQ are mentioned below in brief.

- LINQ to Objects
- LINQ to XML(XLINQ)
- LINQ to DataSet
- LINQ to SQL (DLINQ)
- LINQ to Entities

Apart from the above, there is also a LINQ type named PLINQ which is Microsoft's parallel LINQ.

LINQ Architecture in .NET

LINQ has a 3-layered architecture in which the uppermost layer consists of the language extensions and the bottom layer consists of data sources that are typically objects implementing IEnumerable <T> or IQueryable <T> generic interfaces. The architecture is shown below.



Query Expressions

Query expression is nothing but a LINQ query, expressed in a form similar to that of SQL with query operators like Select, Where and OrderBy. Query expressions usually start with the keyword "From".

To access standard LINQ query operators, the namespace System.Query should be imported by default. These expressions are written within a declarative query syntax which was C# 3.0.

Below is an example to show a complete query operation which consists of data source creation, query expression definition and query execution.

C#

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

namespace Operators
{
   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 + " ");
         }
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
97 92 81
```

Extension Methods

Introduced with .NET 3.5, Extension methods are declared in static classes only and allow inclusion of custom methods to objects to perform some precise query operations to extend a class without being an actual member of that class. These can be overloaded also.

In a nutshell, extension methods are used to translate query expressions into traditional method calls (object-oriented).

Difference between LINQ and Stored Procedure

There is an array of differences existing between LINQ and Stored procedures. These differences are mentioned below.

- Stored procedures are much faster than a LINQ query as they follow an expected execution plan.
- It is easy to avoid run-time errors while executing a LINQ query than in comparison to a stored procedure as the former has Visual Studio's Intellisense support as well as full-type checking during compile-time.
- LINQ allows debugging by making use of .NET debugger which is not in case of stored procedures.
- LINQ offers support for multiple databases in contrast to stored procedures, where it is essential to re-write the code for diverse types of databases.
- Deployment of LINQ based solution is easy and simple in comparison to deployment of a set of stored procedures.

Need for LINQ

Prior to LINQ, it was essential to learn C#, SQL, and various APIs that bind together the both to form a complete application. Since, these data sources and programming languages face an impedance mismatch; a need of short coding is felt.

Below is an example of how many diverse techniques were used by the developers while querying a data before the advent of LINQ.

```
SqlConnection sqlConnection = new SqlConnection(connectString);
SqlConnection.Open();

System.Data.SqlClient.SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Connection = sqlConnection;

sqlCommand.CommandText = "Select * from Customer";
```

```
return sqlCommand.ExecuteReader (CommandBehavior.CloseConnection)
```

Interestingly, out of the featured code lines, query gets defined only by the last two. Using LINQ, the same data query can be written in a readable color-coded form like the following one mentioned below that too in a very less time.

```
Northwind db = new Northwind(@"C:\Data\Northwnd.mdf");
var query = from c in db.Customers select c;
```

Advantages of LINQ

LINQ offers a host of advantages and among them the foremost is its powerful expressiveness which enables developers to express declaratively. Some of the other advantages of LINQ are given below.

- LINQ offers syntax highlighting that proves helpful to find out mistakes during design time.
- LINQ offers IntelliSense which means writing more accurate queries easily.
- Writing codes is quite faster in LINQ and thus development time also gets reduced significantly.
- LINQ makes easy debugging due to its integration in the C# language.
- Viewing relationship between two tables is easy with LINQ due to its hierarchical feature and this enables composing queries joining multiple tables in less time.
- LINQ allows usage of a single LINQ syntax while querying many diverse data sources and this is mainly because of its unitive foundation.
- LINQ is extensible that means it is possible to use knowledge of LINQ to querying new data source types.
- LINQ offers the facility of joining several data sources in a single query as well as breaking complex problems into a set of short queries easy to debug.
- LINQ offers easy transformation for conversion of one data type to another like transforming SQL data to XML data.

2. LINQ – Environment

Before starting with LINQ programs, it is best to first understand the nuances of setting up a LINQ environment. LINQ needs a .NET framework, a revolutionary platform to have a diverse kind of applications. A LINQ query can be written either in C# or Visual Basic conveniently.

Microsoft offers tools for both of these languages i.e. C# and Visual Basic by means of Visual Studio. Our examples are all compiled and written in Visual Studio 2010. However, Visual Basic 2013 edition is also available for use. It is the latest version and has many similarities with Visual Studio 2012.

Getting Visual Studio 2010 Installed on Windows 7

Visual Studio can be installed either from an installation media like a DVD. Administrator credentials are required to install Visual Basic 2010 on your system successfully. It is vital to disconnect all removable USB from the system prior to installation otherwise the installation may get failed. Some of the hardware requirements essential to have for installation are the following ones.

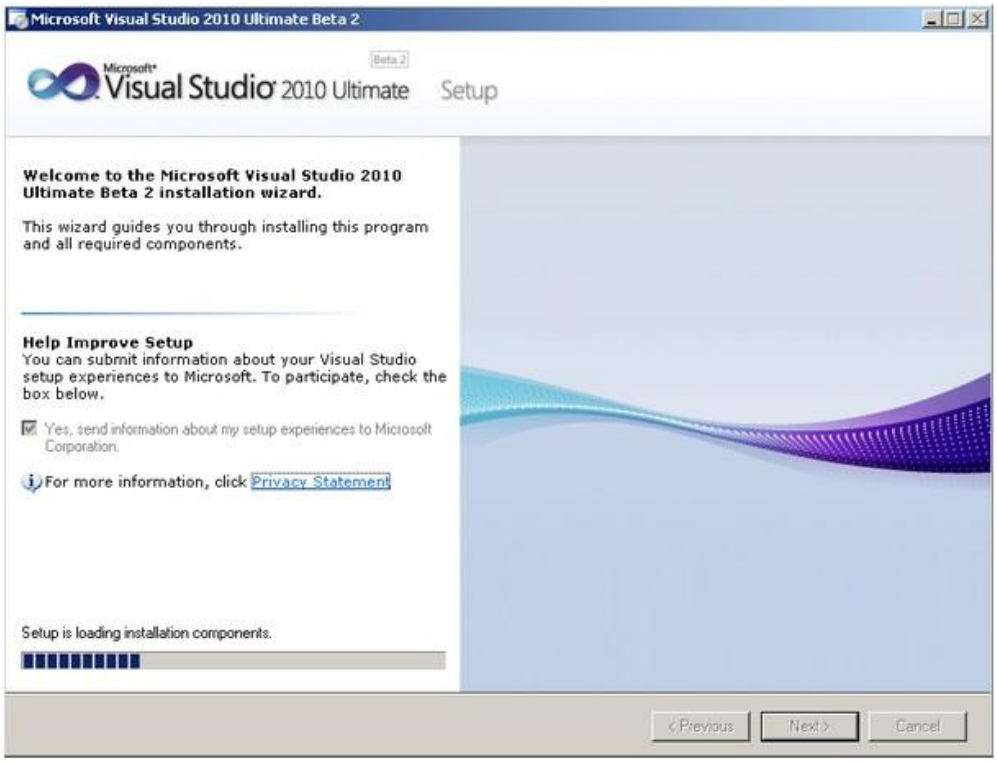
Hardware Requirements

- 1.6 GHz or more
- 1 GB RAM
- 3 GB(Available hard-disk space)
- 5400 RPM hard-disk drive
- DirectX 9 compatible video card
- DVD-ROM drive

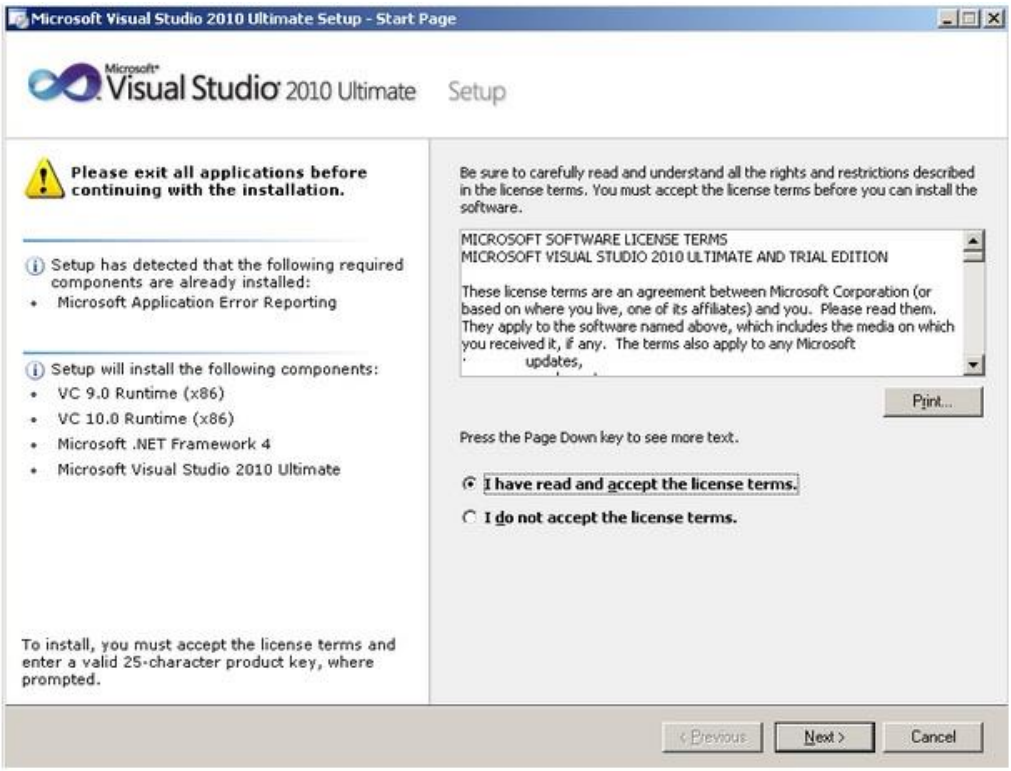
Installation Steps

- **Step 1:** First after inserting the DVD with Visual Studio 2010 Package, click on **Install or run program from your media** appearing in a pop-up box on the screen.
- **Step 2:** Now set up for Visual Studio will appear on the screen. Choose **Install Microsoft Visual Studio 2010**.

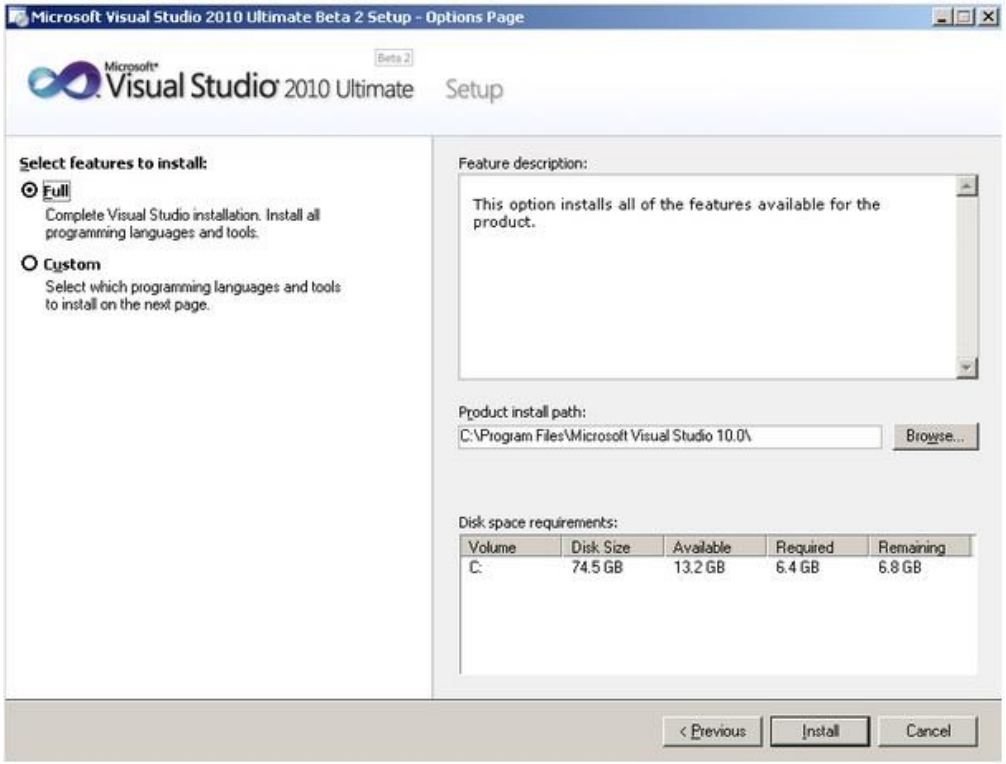
- **Step 3:** As soon as you will click, it the process will get initiated and a set up window will appear on your screen. After completion of loading of the installation components which will take some time, click on **Next** button to move to the next step.



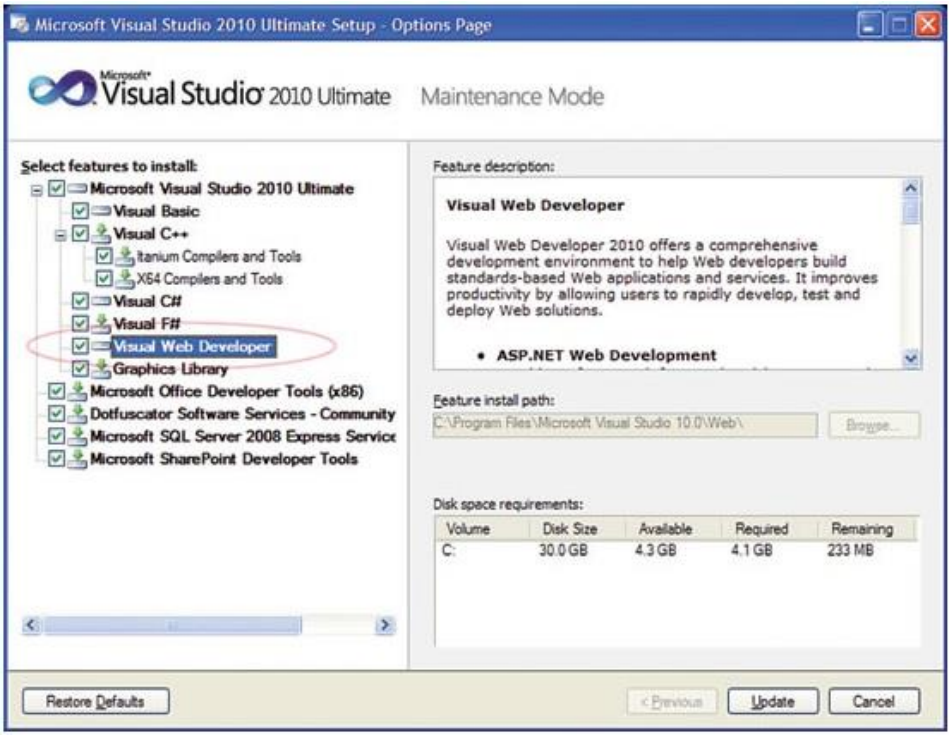
- **Step 4:** This is the last step of installation and a start page will appear in which simply choose "I have read and accept the license terms" and click on **Next** button.



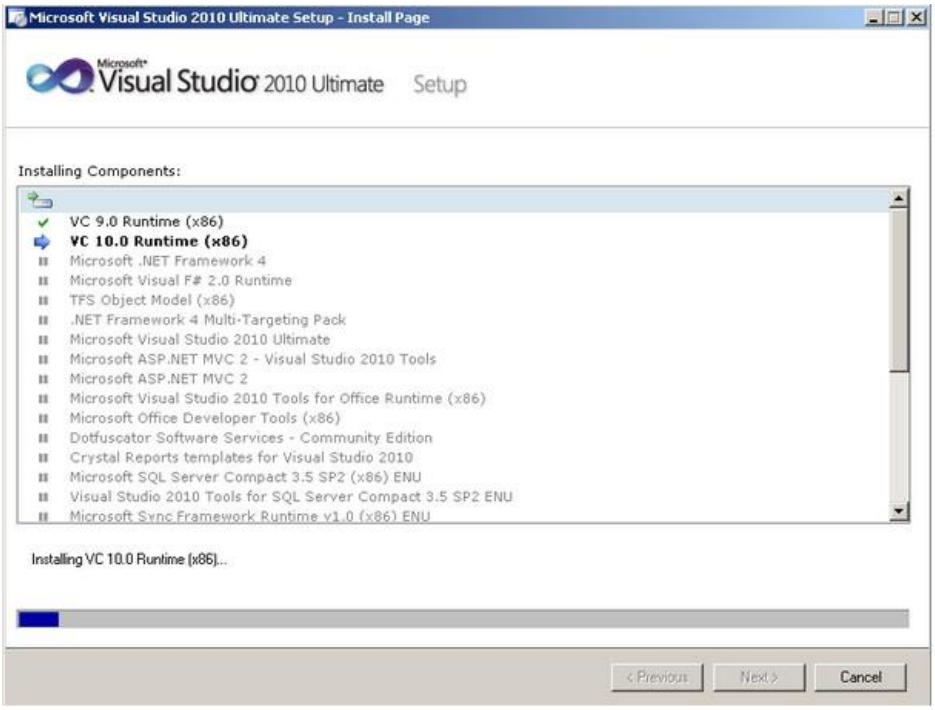
- **Step 5**: Now select features to install from the options page appearing on your screen. You can either choose Full or Custom option. If you have less disk space than required shown in the disk space requirements, then go for Custom.



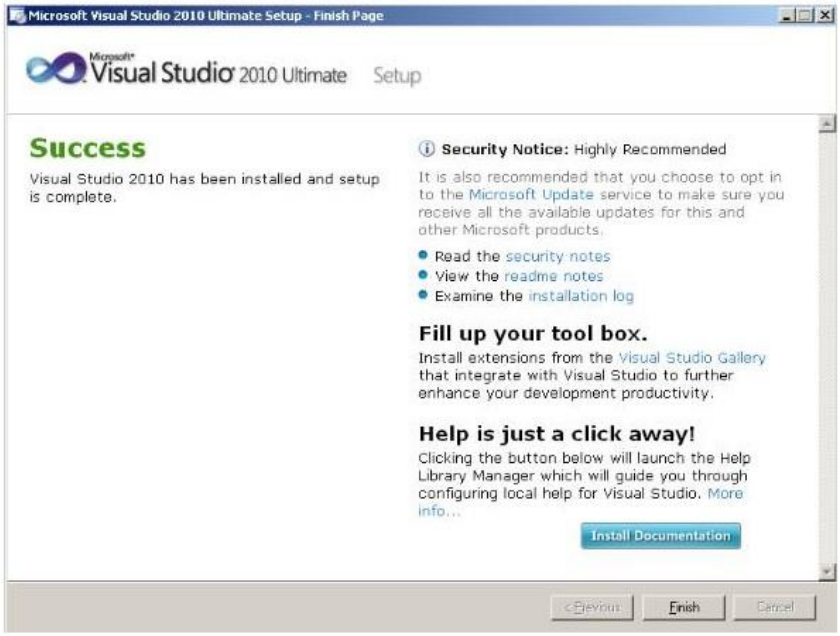
- **Step 6:** When you choose Custom option, the following window will appear. Select the features that you want to install and click **Update** or else go to step 7. However, it is recommended not to go with the custom option as in future, you may need the features you have chosen to not have.



- **Step 7:** Soon a pop up window will be shown and the installation will start which may take a long time. Remember, this is for installing all the components.

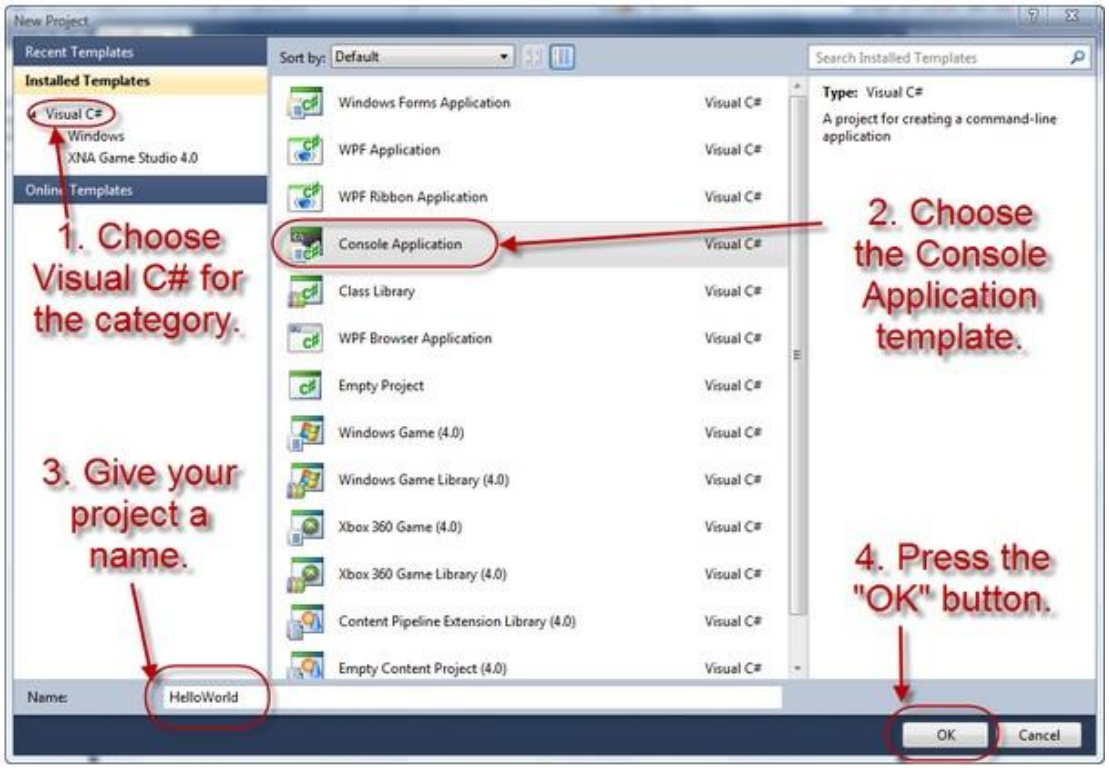


- **Step 8**: Finally, you will be able to view a message in a window that the installation has been completed successfully. Click **Finish.**



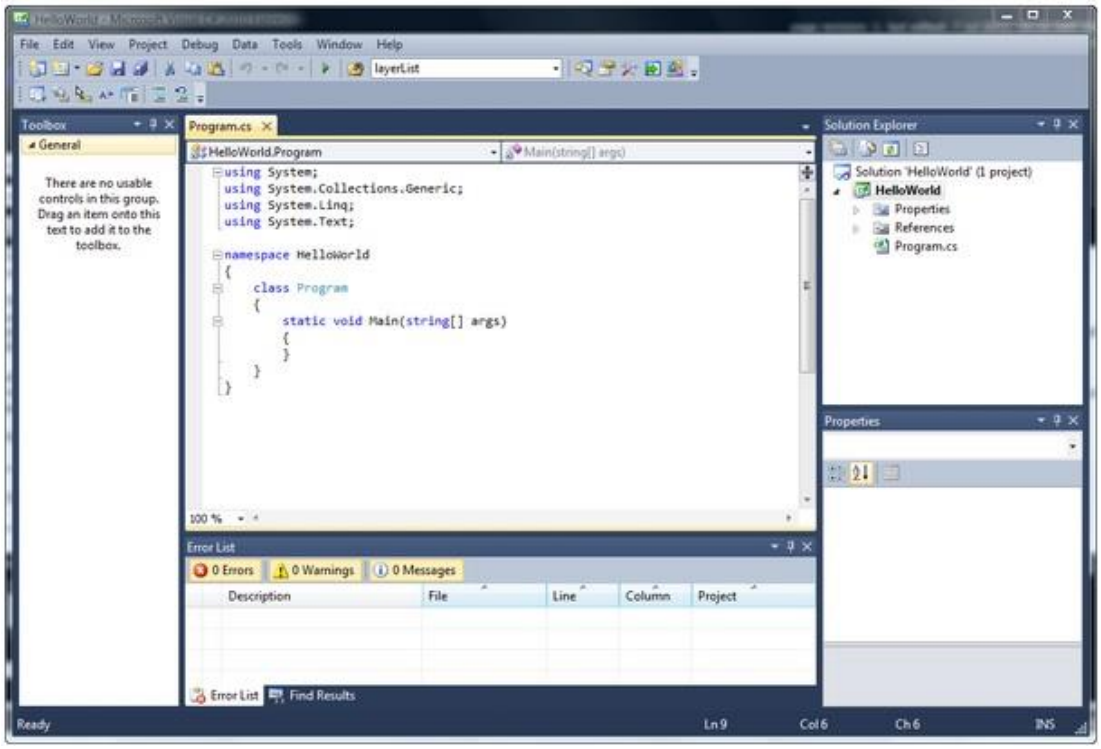
Writing C# Program using LINQ in Visual Studio 2010

1. Start Visual Studio 2010 Ultimate edition and choose File followed by New Project from the menu.
2. A new project dialog box will appear on your screen.
3. Now choose Visual C# as a category under installed templates and next choose Console Application template as shown in figure below.



4. Give a name to your project in the bottom name box and press OK.

5. The new project will appear in the Solution Explorer in the right-hand side of a new dialog box on your screen.



6. Now choose Program.cs from the Solution Explorer and you can view the code in the editor window which starts with 'using System'.
7. Here you can start to code your following C# program.

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

namespace HelloWorld
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Hello World")
         Console.ReadKey();
      }
   }
}
```

8. Press F5 key and run your project. It is highly recommended to save the project by choosing **File → Save All** before running the project.

Writing VB Program using LINQ in Visual Studio 2010

1. Start Visual Studio 2010 Ultimate edition and choose File followed by New Project from the menu.
2. A new project dialog box will appear on your screen.
3. Now chose Visual Basic as a category under installed templates and next choose Console Application template.
4. Give a name to your project in the bottom name box and press OK.
5. You will get a screen with Module1.vb. Start writing your VB code here using LINQ.

```
Module Module1
   Sub Main()
      Console.WriteLine("Hello World")
      Console.ReadLine()
   End Sub
End Module
```

6. Press F5 key and run your project. It is highly recommended to save the project by choosing File → Save All before running the project.

When the above code of C# or VB is compiled and run, it produces the following result:

```
Hello World
```

3. LINQ – Query Operators

A set of extension methods forming a query pattern is known as LINQ Standard Query Operators. As building blocks of LINQ query expressions, these operators offer a range of query capabilities like filtering, sorting, projection, aggregation, etc.

LINQ standard query operators can be categorized into the following ones on the basis of their functionality.

- Filtering Operators
- Join Operators
- Projection Operations
- Sorting Operators
- Grouping Operators
- Conversions
- Concatenation
- Aggregation
- Quantifier Operations
- Partition Operations
- Generation Operations
- Set Operations
- Equality
- Element Operators

Filtering Operators

Filtering is an operation to restrict the result set such that it has only selected elements satisfying a particular condition.

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Where	Filter values based on a predicate function	Where	Where
OfType	Filter values based on their ability to be as a specified type	Not Applicable	Not Applicable

Filtering Operators in LINQ

Filtering is an operation to restrict the result set such that it has only selected elements satisfying a particular condition.

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Where	Filter values based on a predicate function	Where	Where
OfType	Filter values based on their ability to be as a specified type	Not Applicable	Not Applicable

Example of Where – Query Expression

C#

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

namespace Operators
{
   class Program
   {
      static void Main(string[] args)
      {
         string[] words = { "humpty", "dumpty", "set", "on", "a", "wall" };

         IEnumerable<string> query = from word in words where word.Length == 3
select word;

         foreach (string str in query)
            Console.WriteLine(str);
            Console.ReadLine();
      }
   }
}
```

VB

```
Module Module1

   Sub Main()
      Dim words As String() = {"humpty", "dumpty", "set", "on", "a", "wall"}

      Dim query = From word In words Where word.Length = 3 Select word

      For Each n In query
         Console.WriteLine(n)

      Next
         Console.ReadLine()
   End Sub

End Module
```

When the above code in C# or VB is compiled and executed, it produces the following result:

```
set
```

Join Operators

Joining refers to an operation in which data sources with difficult to follow relationships with each other in a direct way are targeted.

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Join	The operator join two sequences on basis of matching keys	join … in … on … equals …	From x In …, y In … Where x.a = y.a
GroupJoin	Join two sequences and group the matching elements	join … in … on … equals … into …	Group Join … In … On …

Join Operators in LINQ

Joining refers to an operation in which data sources with difficult to follow relationships with each other in a direct way are targeted.

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Join	The operator join two sequences on basis of matching keys	join … in … on … equals …	From x In …, y In … Where x.a = y.a
GroupJoin	Join two sequences and group the matching elements	join … in … on … equals … into …	Group Join … In … On …

Example of Join – Query Expression

C#

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

namespace Operators
{
   class JoinTables
   {
      class DepartmentClass
      {
         public int DepartmentId { get; set; }
         public string Name { get; set; }
      }

      class EmployeeClass
      {
         public int EmployeeId { get; set; }
         public string EmployeeName { get; set; }
         public int DepartmentId { get; set; }
      }

      static void Main(string[] args)
      {
         List <DepartmentClass> departments = new List <DepartmentClass>();
```

```
         departments.Add(new DepartmentClass { DepartmentId = 1, Name =
"Account" });
         departments.Add(new DepartmentClass { DepartmentId = 2, Name = "Sales"
});
         departments.Add(new DepartmentClass { DepartmentId = 3, Name =
"Marketing" });


         List <EmployeeClass> employees = new List <EmployeeClass>();
         employees.Add(new EmployeeClass { DepartmentId = 1, EmployeeId = 1,
EmployeeName = "William" });
         employees.Add(new EmployeeClass { DepartmentId = 2, EmployeeId = 2,
EmployeeName = "Miley" });
         employees.Add(new EmployeeClass { DepartmentId = 1, EmployeeId = 3,
EmployeeName = "Benjamin" });



         var list = (from e in employees join d in departments on
e.DepartmentId equals d.DepartmentId select new
            {
               EmployeeName = e.EmployeeName,
               DepartmentName = d.Name
            });


         foreach (var e in list)
         {
            Console.WriteLine("Employee Name = {0} , Department Name = {1}",
e.EmployeeName, e.DepartmentName);
         }


         Console.WriteLine("\nPress any key to continue.");
         Console.ReadKey();
      }
   }
}
```

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**