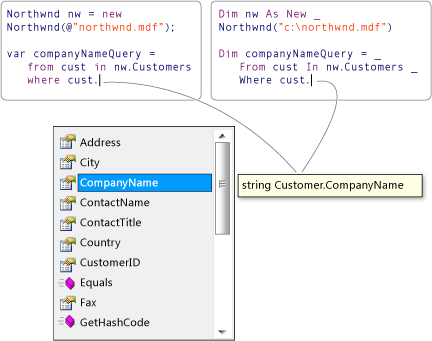
Visual Studio 2010

**Introduction to LINQ**

Language-Integrated Query (LINQ) is an innovation introduced in Visual Studio 2008 and .NET Framework version 3.5 that bridges the gap between the world of objects and the world of data.

Traditionally, queries against data are expressed as simple strings without type checking at compile time or IntelliSense support. 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. LINQ makes a *query* a first-class language construct in C# and Visual Basic. You write queries against strongly typed collections of objects by using language keywords and familiar operators. The following illustration shows a partially-completed LINQ query against a SQL Server database in C# with full type checking and IntelliSense support.



In Visual Studio you can write LINQ queries in Visual Basic or C# with SQL Server databases, XML documents, ADO.NET Datasets, and any collection of objects that supports [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) or the generic [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface. LINQ support for the ADO.NET Entity Framework is also planned, and LINQ providers are being written by third parties for many Web services and other database implementations.

You can use LINQ queries in new projects, or alongside non-LINQ queries in existing projects. The only requirement is that the project target .NET Framework 3.5 or later.

http://i.msdn.microsoft.com/Global/Images/clear.gif Next Steps

To learn more details about LINQ, start by becoming familiar with some basic concepts in the Getting Started section for your language of choice:

* [Getting Started with LINQ in C#](http://msdn.microsoft.com/en-us/library/bb397933.aspx)
* [Getting Started with LINQ in Visual Basic](http://msdn.microsoft.com/en-us/library/bb397910.aspx)

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

* SQL Server databases: [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx)
* XML documents: [LINQ to XML](http://msdn.microsoft.com/en-us/library/bb387098.aspx)
* ADO.NET Datasets: [LINQ to DataSet](http://msdn.microsoft.com/en-us/library/bb386977.aspx)
* .NET collections, files, strings and so on: [LINQ to Objects](http://msdn.microsoft.com/en-us/library/bb397919.aspx)

Visual Studio 2010

**Introduction to LINQ Queries (C#)**

A *query* is an expression that retrieves data from a data source. Queries are usually expressed in a specialized query language. Different languages have been developed over time for the various types of data sources, for example SQL for relational databases and XQuery for XML. Therefore, developers have had to learn a new query language for each type of data source or data format that they must support. LINQ simplifies this situation by offering a consistent model for working with data across various kinds of data sources and formats. In a LINQ query, you are always working with objects. You use the same basic coding patterns to query and transform data in XML documents, SQL databases, ADO.NET Datasets, .NET collections, and any other format for which a LINQ provider is available.

 Three Parts of a Query Operation

All LINQ query operations consist of three distinct actions:

1. Obtain the data source.
2. Create the query.
3. Execute the query.

The following example shows how the three parts of a query operation are expressed in source code. The example uses an integer array as a data source for convenience; however, the same concepts apply to other data sources also. This example is referred to throughout the rest of this topic.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl58_ctl00_ctl00_code');" \o "Copy Code)

class IntroToLINQ

{

static void Main()

{

// The Three Parts of a LINQ Query:

// 1. Data source.

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

// 2. Query creation.

// numQuery is an IEnumerable<int>

var numQuery =

from num in numbers

where (num % 2) == 0

select num;

// 3. Query execution.

foreach (int num in numQuery)

{

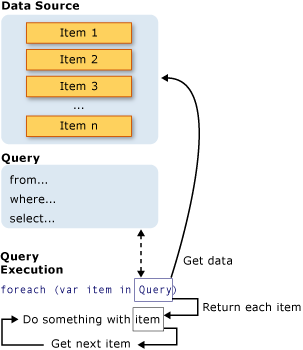
Console.Write("{0,1} ", num);

}

}

}

The following illustration shows the complete query operation. In LINQ the execution of the query is distinct from the query itself; in other words you have not retrieved any data just by creating a query variable.



 The Data Source

In the previous example, because the data source is an array, it implicitly supports the generic [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface. This fact means it can be queried with LINQ. A query is executed in a **foreach** statement, and **foreach** requires [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) or [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx). Types that support [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) or a derived interface such as the generic [IQueryable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/bb351562.aspx) are called *queryable types*.

A queryable type requires no modification or special treatment to serve as a LINQ data source. If the source data is not already in memory as a queryable type, the LINQ provider must represent it as such. For example, LINQ to XML loads an XML document into a queryable [XElement](http://msdn.microsoft.com/en-us/library/system.xml.linq.xelement.aspx) type:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl59_ctl00_ctl06_code');" \o "Copy Code)

// Create a data source from an XML document.

// using System.Xml.Linq;

XElement contacts = XElement.Load(@"c:\myContactList.xml");

With LINQ to SQL, you first create an object-relational mapping at design time either manually or by using the [Object Relational Designer (O/R Designer)](http://msdn.microsoft.com/en-us/library/bb384429.aspx). You write your queries against the objects, and at run-time LINQ to SQL handles the communication with the database. In the following example, Customers represents a specific table in the database, and the type of the query result, [IQueryable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/bb351562.aspx), derives from [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx).

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl59_ctl00_ctl10_code');" \o "Copy Code)

Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// Query for customers in London.

IQueryable<Customer> custQuery =

from cust in db.Customers

where cust.City == "London"

select cust;

For more information about how to create specific types of data sources, see the documentation for the various LINQ providers. However, the basic rule is very simple: a LINQ data source is any object that supports the generic [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface, or an interface that inherits from it.

|  |
| --- |
| **NoteNote** |
| Types such as [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) that support the non-generic [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) interface can also be used as a LINQ data source. For more information, see [How to: Query an ArrayList with LINQ](http://msdn.microsoft.com/en-us/library/bb397937.aspx). |

 The Query

The query specifies what information to retrieve from the data source or sources. Optionally, a query also specifies how that information should be sorted, grouped, and shaped before it is returned. A query is stored in a query variable and initialized with a query expression. To make it easier to write queries, C# has introduced new query syntax.

The query in the previous example returns all the even numbers from the integer array. The query expression contains three clauses: **from**, **where** and **select**. (If you are familiar with SQL, you will have noticed that the ordering of the clauses is reversed from the order in SQL.) The **from** clause specifies the data source, the **where** clause applies the filter, and the **select** clause specifies the type of the returned elements. These and the other query clauses are discussed in detail in the [LINQ Query Expressions (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb397676.aspx) section. For now, the important point is that in LINQ, the query variable itself takes no action and returns no data. It just stores the information that is required to produce the results when the query is executed at some later point. For more information about how queries are constructed behind the scenes, see [Standard Query Operators Overview](http://msdn.microsoft.com/en-us/library/bb397896.aspx).

|  |
| --- |
| **NoteNote** |
| Queries can also be expressed by using method syntax. For more information, see [LINQ Query Syntax versus Method Syntax (C#)](http://msdn.microsoft.com/en-us/library/bb397947.aspx). |

 Query Execution

**Deferred Execution**

As stated previously, the query variable itself only stores the query commands. The actual execution of the query is deferred until you iterate over the query variable in a **foreach** statement. This concept is referred to as *deferred execution* and is demonstrated in the following example:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl00_code');" \o "Copy Code)

// Query execution.

foreach (int num in numQuery)

{

Console.Write("{0,1} ", num);

}

The **foreach** statement is also where the query results are retrieved. For example, in the previous query, the iteration variable num holds each value (one at a time) in the returned sequence.

Because the query variable itself never holds the query results, you can execute it as often as you like. For example, you may have a database that is being updated continually by a separate application. In your application, you could create one query that retrieves the latest data, and you could execute it repeatedly at some interval to retrieve different results every time.

**Forcing Immediate Execution**

Queries that perform aggregation functions over a range of source elements must first iterate over those elements. Examples of such queries are **Count**, **Max**, **Average**, and **First**. These execute without an explicit **foreach** statement because the query itself must use **foreach** in order to return a result. Note also that these types of queries return a single value, not an **IEnumerable** collection. The following query returns a count of the even numbers in the source array:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl01_code');" \o "Copy Code)

var evenNumQuery =

from num in numbers

where (num % 2) == 0

select num;

int evenNumCount = evenNumQuery.Count();

To force immediate execution of any query and cache its results, you can call the [ToList<(Of <(TSource>)>)](http://msdn.microsoft.com/en-us/library/bb342261.aspx) or [ToArray<(Of <(TSource>)>)](http://msdn.microsoft.com/en-us/library/bb298736.aspx) methods.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl61_ctl00_ctl04_code');" \o "Copy Code)

List<int> numQuery2 =

(from num in numbers

where (num % 2) == 0

select num).ToList();

// or like this:

// numQuery3 is still an int[]

var numQuery3 =

(from num in numbers

where (num % 2) == 0

select num).ToArray();

You can also force execution by putting the **foreach** loop immediately after the query expression. However, by calling **ToList** or **ToArray** you also cache all the data in a single collection object.

Visual Studio 2010

**Visual Basic Features That Support LINQ**

Updated: May 2011

The name Language-Integrated Query (LINQ) refers to technology in Visual Basic that supports query syntax and other language constructs directly in the language. With LINQ, you do not have to learn a new language to query against an external data source. You can query against data in relational databases, XML stores, or objects by using Visual Basic. This integration of query capabilities into the language enables compile-time checking for syntax errors and type safety. This integration also ensures that you already know most of what you have to know to write rich, varied queries in Visual Basic.

The following sections describe the language constructs that support LINQ in enough detail to enable you to get started in reading the introductory documentation, code examples, and sample applications. You can also click the links to find more detailed explanations of how the language features come together to enable language-integrated query. A good place to start is [Walkthrough: Writing Queries in Visual Basic](http://msdn.microsoft.com/en-us/library/bb385164.aspx).

 Query Expressions

Query expressions in Visual Basic can be expressed in a declarative syntax similar to that of SQL or XQuery. At compile time, query syntax is converted into method calls to a LINQ provider's implementation of the standard query operator extension methods. Applications control which standard query operators are in scope by specifying the appropriate namespace with an **Imports** statement. Syntax for a Visual Basic query expression looks like this:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl47_ctl00_ctl00_code');" \o "Copy Code)

Dim londonCusts = From cust In customers

Where cust.City = "London"

Order By cust.Name Ascending

Select cust.Name, cust.Phone

For more information, see [Introduction to LINQ in Visual Basic](http://msdn.microsoft.com/en-us/library/bb763068.aspx).

 Implicitly Typed Variables

Instead of explicitly specifying a type when you declare and initialize a variable, you can enable the compiler to infer and assign the type. This is referred to as *local type inference*.

Variables whose types are inferred are strongly typed, just like variables whose type you specify explicitly. Local type inference works only when you are defining a local variable inside a method body. For more information, see [Option Infer Statement](http://msdn.microsoft.com/en-us/library/bb384665.aspx) and [Local Type Inference (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb384937.aspx).

The following example illustrates local type inference. To use this example, you must set **Option Infer** to **On**.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl02_code');" \o "Copy Code)

' The variable aNumber will be typed as an integer.

Dim aNumber = 5

' The variable aName will be typed as a String.

Dim aName = "Virginia"

Local type inference also makes it possible to create anonymous types, which are described later in this topic and are necessary for LINQ queries.

In the following LINQ example, type inference occurs if **Option Infer** is either **On** or **Off**. A compile-time error occurs if **Option Infer** is **Off** and **Option Strict** is **On**.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl03_code');" \o "Copy Code)

' Query example.

' If numbers is a one-dimensional array of integers, num will be typed

' as an integer and numQuery will be typed as IEnumerable(Of Integer)--

' basically a collection of integers.

Dim numQuery = From num In numbers

Where num Mod 2 = 0

Select num

 Object Initializers

Object initializers are used in query expressions when you have to create an anonymous type to hold the results of a query. They also can be used to initialize objects of named types outside of queries. By using an object initializer, you can initialize an object in a single line without explicitly calling a constructor. Assuming that you have a class named Customer that has public Name and Phone properties, along with other properties, an object initializer can be used in this manner:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl49_ctl00_ctl00_code');" \o "Copy Code)

Dim aCust = New Customer With {.Name = "Mike",

.Phone = "555-0212"}

For more information, see [Object Initializers: Named and Anonymous Types (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb385125.aspx).

 Anonymous Types

Anonymous types provide a convenient way to temporarily group a set of properties into an element that you want to include in a query result. This enables you to choose any combination of available fields in the query, in any order, without defining a named data type for the element.

An *anonymous type* is constructed dynamically by the compiler. The name of the type is assigned by the compiler, and it might change with each new compilation. Therefore, the name cannot be used directly. Anonymous types are initialized in the following way:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl50_ctl00_ctl00_code');" \o "Copy Code)

' Outside a query.

Dim product = New With {.Name = "paperclips", .Price = 1.29}

' Inside a query.

' You can use the existing member names of the selected fields, as was

' shown previously in the Query Expressions section of this topic.

Dim londonCusts1 = From cust In customers

Where cust.City = "London"

Select cust.Name, cust.Phone

' Or you can specify new names for the selected fields.

Dim londonCusts2 = From cust In customers

Where cust.City = "London"

Select CustomerName = cust.Name,

CustomerPhone = cust.Phone

For more information, see [Anonymous Types (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb384767.aspx).

 Extension Methods

Extension methods enable you to add methods to a data type or interface from outside the definition. This feature enables you to, in effect, add new methods to an existing type without actually modifying the type. The standard query operators are themselves a set of extension methods that provide LINQ query functionality for any type that implements [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx). Other extensions to [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) include [Count](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.count.aspx), [Union](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.union.aspx), and [Intersect](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.intersect.aspx).

The following extension method adds a print method to the [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) class.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl06_code');" \o "Copy Code)

' Import System.Runtime.CompilerServices to use the Extension attribute.

<Extension()>

Public Sub Print(ByVal str As String)

Console.WriteLine(str)

End Sub

The method is called like an ordinary instance method of [String](http://msdn.microsoft.com/en-us/library/system.string.aspx):

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl08_code');" \o "Copy Code)

Dim greeting As String = "Hello"

greeting.Print()

For more information, see [Extension Methods (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb384936.aspx).

 Lambda Expressions

A lambda expression is a function without a name that calculates and returns a single value. Unlike named functions, a lambda expression can be defined and executed at the same time. The following example displays 4.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl00_code');" \o "Copy Code)

Console.WriteLine((Function(num As Integer) num + 1)(3))

You can assign the lambda expression definition to a variable name and then use the name to call the function. The following example also displays 4.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl01_code');" \o "Copy Code)

Dim add1 = Function(num As Integer) num + 1

Console.WriteLine(add1(3))

In LINQ, lambda expressions underlie many of the standard query operators. The compiler creates lambda expressions to capture the calculations that are defined in fundamental query methods such as **Where**, **Select**, **Order By**, **Take While**, and others.

For example, the following code defines a query that returns all senior students from a list of students.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl02_code');" \o "Copy Code)

Dim seniorsQuery = From stdnt In students

Where stdnt.Year = "Senior"

Select stdnt

The query definition is compiled into code that is similar to the following example, which uses two lambda expressions to specify the arguments for **Where** and **Select**.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl03_code');" \o "Copy Code)

Dim seniorsQuery2 = students.

Where(Function(st) st.Year = "Senior").

Select(Function(s) s)

Either version can be run by using a **For Each** loop:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl04_code');" \o "Copy Code)

For Each senior In seniorsQuery

Console.WriteLine(senior.Last & ", " & senior.First)

Next

LINQ to SQL

**Getting Started (LINQ to SQL)**

By using LINQ to SQL, you can use the LINQ technology to access SQL databases just as you would access an in-memory collection.

For example, the nw object in the following code is created to represent the Northwind database, the Customers table is targeted, the rows are filtered for Customers from London, and a string for CompanyName is selected for retrieval.

When the loop is executed, the collection of CompanyName values is retrieved.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

' Northwnd inherits from System.Data.Linq.DataContext.

Dim nw As New Northwnd("c:\northwnd.mdf")

' or, if you are not using SQL Server Express

' Dim nw As New Northwnd("Database=Northwind;Server=dschwart7;Integrated Security=SSPI")

Dim companyNameQuery = \_

From cust In nw.Customers \_

Where cust.City = "London" \_

Select cust.CompanyName

For Each customer In companyNameQuery

Console.WriteLine(customer)

Next

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl02_code');" \o "Copy Code)

// Northwnd inherits from System.Data.Linq.DataContext.

Northwnd nw = new Northwnd(@"northwnd.mdf");

// or, if you are not using SQL Server Express

// Northwnd nw = new Northwnd("Database=Northwind;Server=server\_name;Integrated Security=SSPI");

var companyNameQuery =

from cust in nw.Customers

where cust.City == "London"

select cust.CompanyName;

foreach (var customer in companyNameQuery)

{

Console.WriteLine(customer);

}

 Next Steps

For some additional examples, including inserting and updating, see [What You Can Do With LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb882643.aspx).

Next, try some walkthroughs and tutorials to have a hands-on experience of using LINQ to SQL. See [Learning by Walkthroughs (LINQ to SQL)](http://msdn.microsoft.com/en-us/library/bb399349.aspx).

Finally, learn how to get started on your own LINQ to SQL project by reading [Typical Steps for Using LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb387007.aspx).

Language-Integrated Query (LINQ)

**LINQ to XML Overview**

XML has been widely adopted as a way to format data in many contexts. For example, you can find XML on the Web, in configuration files, in Microsoft Office Word files, and in databases.

LINQ to XML is an up-to-date, redesigned approach to programming with XML. It provides the in-memory document modification capabilities of the Document Object Model (DOM), and supports LINQ query expressions. Although these query expressions are syntactically different from XPath, they provide similar functionality.

 LINQ to XML Developers

LINQ to XML targets a variety of developers. For an average developer who just wants to get something done, LINQ to XML makes XML easier by providing a query experience that is similar to SQL. With just a bit of study, programmers can learn to write succinct and powerful queries in their programming language of choice.

Professional developers can use LINQ to XML to greatly increase their productivity. With LINQ to XML, they can write less code that is more expressive, more compact, and more powerful. They can use query expressions from multiple data domains at the same time.

 What Is LINQ to XML?

LINQ to XML is a LINQ-enabled, in-memory XML programming interface that enables you to work with XML from within the .NET Framework programming languages.

LINQ to XML is like the Document Object Model (DOM) in that it brings the XML document into memory. 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. However, LINQ to XML differs from DOM: It provides a new object model that is lighter weight and easier to work with, and that takes advantage of language improvements in Visual C# 2008.

The most important advantage of LINQ to XML is its integration with Language-Integrated Query (LINQ). This integration enables you to write queries on the in-memory XML document to retrieve collections of elements and attributes. The query capability of LINQ to XML is comparable in functionality (although not in syntax) to XPath and XQuery. The integration of LINQ in Visual C# 2008 provides stronger typing, compile-time checking, and improved debugger support.

Another advantage of LINQ to XML is the ability to use query results as parameters to [XElement](http://msdn.microsoft.com/en-us/library/system.xml.linq.xelement.aspx) and [XAttribute](http://msdn.microsoft.com/en-us/library/system.xml.linq.xattribute.aspx) object constructors enables a powerful approach to creating XML trees. This approach, called *functional construction*, enables developers to easily transform XML trees from one shape to another.

For example, you might have a typical XML purchase order as described in [Sample XML File: Typical Purchase Order (LINQ to XML)](http://msdn.microsoft.com/en-us/library/bb387012.aspx). 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:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl03_code');" \o "Copy Code)

IEnumerable<string> partNos =

from item in purchaseOrder.Descendants("Item")

select (string) item.Attribute("PartNumber");

In Visual Basic, the same query can be written as follows:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl04_code');" \o "Copy Code)

Dim partNos = \_

From item In purchaseOrder...<Item> \_

Select item.@PartNumber

As another example, you might want a list, sorted by part number, of the items with a value greater than $100. To obtain this information, you could run the following query:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl05_code');" \o "Copy Code)

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;

In Visual Basic, the same query can be written as follows:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl06_code');" \o "Copy Code)

Dim partNos = \_

From item In purchaseOrder...<Item> \_

Where (item.<Quantity>.Value \* \_

item.<USPrice>.Value) > 100 \_

Order By item.<PartNumber>.Value \_

Select item

In addition to these LINQ capabilities, LINQ to XML provides an improved XML programming interface. Using LINQ to XML, you can:

* Load XML from files or streams.
* Serialize XML to files or streams.
* Create XML from scratch by using functional construction.
* Query XML using XPath-like axes.
* Manipulate the in-memory XML tree by using methods such as [Add](http://msdn.microsoft.com/en-us/library/system.xml.linq.xcontainer.add.aspx), [Remove](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.remove.aspx), [ReplaceWith](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.replacewith.aspx), and [SetValue](http://msdn.microsoft.com/en-us/library/system.xml.linq.xelement.setvalue.aspx).
* Validate XML trees using XSD.
* Use a combination of these features to transform XML trees from one shape into another.

 Creating XML Trees

IOne of the most significant advantages of programming with LINQ to XML is that it is easy to create XML trees. For example, to create a small XML tree, you can write C# code as follows:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl00_code');" \o "Copy Code)

XElement contacts =

new XElement("Contacts",

new XElement("Contact",

new XElement("Name", "Patrick Hines"),

new XElement("Phone", "206-555-0144",

new XAttribute("Type", "Home")),

new XElement("phone", "425-555-0145",

new XAttribute("Type", "Work")),

new XElement("Address",

new XElement("Street1", "123 Main St"),

new XElement("City", "Mercer Island"),

new XElement("State", "WA"),

new XElement("Postal", "68042")

)

)

);

In Visual Basic, the code to construct the XML tree is even simpler, because it uses an XML literal:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl01_code');" \o "Copy Code)

Dim contacts = \_

<Contacts>

<Contact>

<Name>Patrick Hines</Name>

<Phone Type="Home">206-555-0144</Phone>

<Phone Type="Work">425-555-0145</Phone>

<Address>

<Street1>123 Main St</Street1>

<City>Mercer Island</City>

<State>WA</State>

<Postal>68042</Postal>

</Address>

</Contact>

</Contacts>

The Visual Basic compiler translates XML literals into LINQ to XML method calls.

For more information, see [Creating XML Trees](http://msdn.microsoft.com/en-us/library/bb387068.aspx).

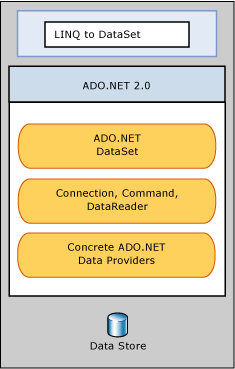
.NET Framework 4

**LINQ to DataSet**

LINQ to DataSet makes it easier and faster to query over data cached in a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object. Specifically, LINQ to DataSet simplifies querying by enabling developers to write queries from the programming language itself, instead of by using a separate query language. This is especially useful for Visual Studio developers, who can now take advantage of the compile-time syntax checking, static typing, and IntelliSense support provided by the Visual Studio in their queries.

LINQ to DataSet can also be used to query over data that has been consolidated from one or more data sources. This enables many scenarios that require flexibility in how data is represented and handled, such as querying locally aggregated data and middle-tier caching in Web applications. In particular, generic reporting, analysis, and business intelligence applications require this method of manipulation.

The LINQ to DataSet functionality is exposed primarily through the extension methods in the [DataRowExtensions](http://msdn.microsoft.com/en-us/library/system.data.datarowextensions.aspx) and [DataTableExtensions](http://msdn.microsoft.com/en-us/library/system.data.datatableextensions.aspx) classes. LINQ to DataSet builds on and uses the existing ADO.NET 2.0 architecture, and is not meant to replace ADO.NET 2.0 in application code. Existing ADO.NET 2.0 code will continue to function in a LINQ to DataSet application. The relationship of LINQ to DataSet to ADO.NET 2.0 and the data store is illustrated in the following diagram.



.NET Framework 4

**LINQ to DataSet Overview**

The [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is one of the more widely used components of ADO.NET. It is a key element of the disconnected programming model that ADO.NET is based on, and it enables you to explicitly cache data from different data sources. For the presentation tier, the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is tightly integrated with GUI controls for data-binding. For the middle-tier, it provides a cache that preserves the relational shape of data, and includes fast simple query and hierarchy navigation services. A common technique used to lower the number of requests on a database is to use the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) for caching in the middle-tier. For example, consider a data-driven ASP.NET Web application. Often, a significant portion of the application data does not change frequently and is common across sessions or users. This data can be kept in memory on the Web server, which reduces the number of requests against the database and speeds up the user’s interactions. Another useful aspect of the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is that it allows an application to bring subsets of data from one or more data source into the application space. The application can then manipulate the data in-memory, while retaining its relational shape.

Despite its prominence, the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) has limited query capabilities. The [Select](http://msdn.microsoft.com/en-us/library/system.data.datatable.select.aspx) method can be used for filtering and sorting, and the [GetChildRows](http://msdn.microsoft.com/en-us/library/system.data.datarow.getchildrows.aspx) and [GetParentRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.getparentrow.aspx) methods can be used for hierarchy navigation. For anything more complex, however, the developer must write a custom query. This can result in applications that perform poorly and are difficult to maintain.

LINQ to DataSet makes it easier and faster to query over data cached in a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object. These queries are expressed in the programming language itself, rather than as string literals embedded in the application code. This means that developers do not have to learn a separate query language. Additionally, LINQ to DataSet enables Visual Studio developers to work more productively, because the Visual Studio IDE provides compile-time syntax checking, static typing, and IntelliSense support for LINQ. LINQ to DataSet can also be used to query over data that has been consolidated from one or more data sources. This enables many scenarios that require flexibility in how data is represented and handled. In particular, generic reporting, analysis, and business intelligence applications require this method of manipulation.

 Querying DataSets Using LINQ to DataSet

Before you can begin querying a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object using LINQ to DataSet, you must populate the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). There are several ways to load data into a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx), such as using the [DataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dataadapter.aspx) class or [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx). After the data has been loaded into a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object, you can begin to query it. Formulating queries using LINQ to DataSet is similar to using Language-Integrated Query (LINQ) against other LINQ-enabled data sources. LINQ queries can be performed against single tables in a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) or against more than one table by using the [Join](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.join.aspx) and [GroupJoin](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.groupjoin.aspx) standard query operators.

LINQ queries are supported against both typed and untyped [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) objects. If the schema of the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is known at application design time, a typed [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is recommended. In a typed [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx), the tables and rows have typed members for each of the columns, which makes queries simpler and more readable.

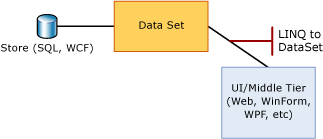
In addition to the standard query operators implemented in System.Core.dll, LINQ to DataSet adds several [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx)-specific extensions that make it easier to query over a set of [DataRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx) objects. These [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx)-specific extensions include operators for comparing sequences of rows, as well as methods that provide access to the column values of a [DataRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx).

 N-tier Applications and LINQ to DataSet

N-tier data applications are data-centric applications that are separated into multiple logical layers (or tiers). A typical N-tier application includes a presentation tier, a middle tier, and a data tier. Separating application components into separate tiers increases the maintainability and scalability of the application. For more information about N-tier data applications, see [Working with Datasets in N-Tier Applications](http://msdn.microsoft.com/en-us/library/bb384587.aspx).

In N-tier applications, the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is often used in the middle-tier to cache information for a Web application. The LINQ to DataSet querying functionality is implemented through extension methods and extends the existing ADO.NET 2.0 [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx).

The following diagram shows how LINQ to DataSet relates to the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) and fits into an n-tier application:



**Loading Data Into a DataSet**

A [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object must first be populated before you can query over it with LINQ to DataSet. There are several different ways to populate the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). For example, you can use LINQ to SQL to query the database and load the results into the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). For more information, see [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx).

Another common way to load data into a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is to use the [DataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dataadapter.aspx) class to retrieve data from the database. This is illustrated in the following example.

 Example

This example uses a [DataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dataadapter.aspx) to query the AdventureWorks database for sales information from the year 2002, and loads the results into a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). After the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) has been populated, you can write queries against it by using LINQ to DataSet. The **FillDataSet** method in this example is used in the example queries in [LINQ to DataSet Examples](http://msdn.microsoft.com/en-us/library/bb399401.aspx). For more information, see [Querying DataSets (LINQ to DataSet)](http://msdn.microsoft.com/en-us/library/bb399358.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl05_code');" \o "Copy Code)

Try

Dim connectionString As String

connectionString = "Data Source=localhost;Initial Catalog=AdventureWorks;" & \_

"Integrated Security=true;"

' Create a new adapter and give it a query to fetch sales order, contact,

' address, and product information for sales in the year 2002. Point connection

' information to the configuration setting "AdventureWorks".

Dim da = New SqlDataAdapter( \_

"SELECT SalesOrderID, ContactID, OrderDate, OnlineOrderFlag, " & \_

"TotalDue, SalesOrderNumber, Status, ShipToAddressID, BillToAddressID " & \_

"FROM Sales.SalesOrderHeader " & \_

"WHERE DATEPART(YEAR, OrderDate) = @year; " & \_

"SELECT d.SalesOrderID, d.SalesOrderDetailID, d.OrderQty, " & \_

"d.ProductID, d.UnitPrice " & \_

"FROM Sales.SalesOrderDetail d " & \_

"INNER JOIN Sales.SalesOrderHeader h " & \_

"ON d.SalesOrderID = h.SalesOrderID " & \_

"WHERE DATEPART(YEAR, OrderDate) = @year; " & \_

"SELECT p.ProductID, p.Name, p.ProductNumber, p.MakeFlag, " & \_

"p.Color, p.ListPrice, p.Size, p.Class, p.Style " & \_

"FROM Production.Product p; " & \_

"SELECT DISTINCT a.AddressID, a.AddressLine1, a.AddressLine2, " & \_

"a.City, a.StateProvinceID, a.PostalCode " & \_

"FROM Person.Address a " & \_

"INNER JOIN Sales.SalesOrderHeader h " & \_

"ON a.AddressID = h.ShipToAddressID OR a.AddressID = h.BillToAddressID " & \_

"WHERE DATEPART(YEAR, OrderDate) = @year; " & \_

"SELECT DISTINCT c.ContactID, c.Title, c.FirstName, " & \_

"c.LastName, c.EmailAddress, c.Phone " & \_

"FROM Person.Contact c " & \_

"INNER JOIN Sales.SalesOrderHeader h " & \_

"ON c.ContactID = h.ContactID " & \_

"WHERE DATEPART(YEAR, OrderDate) = @year;", \_

connectionString)

' Add table mappings.

da.SelectCommand.Parameters.AddWithValue("@year", 2002)

da.TableMappings.Add("Table", "SalesOrderHeader")

da.TableMappings.Add("Table1", "SalesOrderDetail")

da.TableMappings.Add("Table2", "Product")

da.TableMappings.Add("Table3", "Address")

da.TableMappings.Add("Table4", "Contact")

da.Fill(ds)

' Add data relations.

Dim orderHeader As DataTable = ds.Tables("SalesOrderHeader")

Dim orderDetail As DataTable = ds.Tables("SalesOrderDetail")

Dim co As DataRelation = New DataRelation("SalesOrderHeaderDetail", \_

orderHeader.Columns("SalesOrderID"), \_

orderDetail.Columns("SalesOrderID"), True)

ds.Relations.Add(co)

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

Dim orderContact As DataRelation = New DataRelation("SalesOrderContact", \_

contact.Columns("ContactID"), \_

orderHeader.Columns("ContactID"), True)

ds.Relations.Add(orderContact)

Catch ex As SqlException

Console.WriteLine("SQL exception occurred: " & ex.Message)

End Try

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl06_code');" \o "Copy Code)

try

{

// Create a new adapter and give it a query to fetch sales order, contact,

// address, and product information for sales in the year 2002. Point connection

// information to the configuration setting "AdventureWorks".

string connectionString = "Data Source=localhost;Initial Catalog=AdventureWorks;"

+ "Integrated Security=true;";

SqlDataAdapter da = new SqlDataAdapter(

"SELECT SalesOrderID, ContactID, OrderDate, OnlineOrderFlag, " +

"TotalDue, SalesOrderNumber, Status, ShipToAddressID, BillToAddressID " +

"FROM Sales.SalesOrderHeader " +

"WHERE DATEPART(YEAR, OrderDate) = @year; " +

"SELECT d.SalesOrderID, d.SalesOrderDetailID, d.OrderQty, " +

"d.ProductID, d.UnitPrice " +

"FROM Sales.SalesOrderDetail d " +

"INNER JOIN Sales.SalesOrderHeader h " +

"ON d.SalesOrderID = h.SalesOrderID " +

"WHERE DATEPART(YEAR, OrderDate) = @year; " +

"SELECT p.ProductID, p.Name, p.ProductNumber, p.MakeFlag, " +

"p.Color, p.ListPrice, p.Size, p.Class, p.Style, p.Weight " +

"FROM Production.Product p; " +

"SELECT DISTINCT a.AddressID, a.AddressLine1, a.AddressLine2, " +

"a.City, a.StateProvinceID, a.PostalCode " +

"FROM Person.Address a " +

"INNER JOIN Sales.SalesOrderHeader h " +

"ON a.AddressID = h.ShipToAddressID OR a.AddressID = h.BillToAddressID " +

"WHERE DATEPART(YEAR, OrderDate) = @year; " +

"SELECT DISTINCT c.ContactID, c.Title, c.FirstName, " +

"c.LastName, c.EmailAddress, c.Phone " +

"FROM Person.Contact c " +

"INNER JOIN Sales.SalesOrderHeader h " +

"ON c.ContactID = h.ContactID " +

"WHERE DATEPART(YEAR, OrderDate) = @year;",

connectionString);

// Add table mappings.

da.SelectCommand.Parameters.AddWithValue("@year", 2002);

da.TableMappings.Add("Table", "SalesOrderHeader");

da.TableMappings.Add("Table1", "SalesOrderDetail");

da.TableMappings.Add("Table2", "Product");

da.TableMappings.Add("Table3", "Address");

da.TableMappings.Add("Table4", "Contact");

// Fill the DataSet.

da.Fill(ds);

// Add data relations.

DataTable orderHeader = ds.Tables["SalesOrderHeader"];

DataTable orderDetail = ds.Tables["SalesOrderDetail"];

DataRelation order = new DataRelation("SalesOrderHeaderDetail",

orderHeader.Columns["SalesOrderID"],

orderDetail.Columns["SalesOrderID"], true);

ds.Relations.Add(order);

DataTable contact = ds.Tables["Contact"];

DataTable orderHeader2 = ds.Tables["SalesOrderHeader"];

DataRelation orderContact = new DataRelation("SalesOrderContact",

contact.Columns["ContactID"],

orderHeader2.Columns["ContactID"], true);

ds.Relations.Add(orderContact);

}

catch (SqlException ex)

{

Console.WriteLine("SQL exception occurred: " + ex.Message);

}

Visual Studio 2010

**LINQ to Objects**

The term "LINQ to Objects" refers to the use of LINQ queries with any [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) or [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) collection directly, without the use of an intermediate LINQ provider or API such as [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx) or [LINQ to XML](http://msdn.microsoft.com/en-us/library/bb387098.aspx). You can use LINQ to query any enumerable collections such as [List<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx), [Array](http://msdn.microsoft.com/en-us/library/system.array.aspx), or [Dictionary<(Of <(TKey, TValue>)>)](http://msdn.microsoft.com/en-us/library/xfhwa508.aspx). The collection may be user-defined or may be returned by a .NET Framework API.

In a basic sense, LINQ to Objects represents a new approach to collections. In the old way, you had to write complex **foreach** loops that specified how to retrieve data from a collection. In the LINQ approach, you write declarative code that describes what you want to retrieve.

In addition, LINQ queries offer three main advantages over traditional **foreach** loops:

1. They are more concise and readable, especially when filtering multiple conditions.
2. They provide powerful filtering, ordering, and grouping capabilities with a minimum of application code.
3. They can be ported to other data sources with little or no modification.

In general, the more complex the operation you want to perform on the data, the more benefit you will realize by using LINQ instead of traditional iteration techniques.

The purpose of this section is to demonstrate the LINQ approach with some select examples. It is not intended to be exhaustive.

Visual Studio 2010

**How to: Query an ArrayList with LINQ**

When using LINQ to query non-generic [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) collections such as [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx), you must explicitly declare the type of the range variable to reflect the specific type of the objects in the collection. For example, if you have an [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) of Student objects, your [from clause](http://msdn.microsoft.com/en-us/library/bb383978.aspx) (C#) or [From Clause (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb384802.aspx) should look like this:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl06_code');" \o "Copy Code)

// C#

var query = from Student s in arrList

...

'Visual Basic

Dim query = From student As Student In arrList

...

By specifying the type of the range variable, you are casting each item in the [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) to a Student.

The use of an explicitly typed range variable in a query expression is equivalent to calling the [Cast<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb341406.aspx) method. [Cast<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb341406.aspx) throws an exception if the specified cast cannot be performed. [Cast<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb341406.aspx) and [OfType<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb360913.aspx) are the two Standard Query Operator methods that operate on non-generic [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) types. In Visual Basic, you must explicitly call the [Cast<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb341406.aspx) method on the data source to ensure a specific range variable type. For more information, see[Type Relationships in Query Operations (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb384935.aspx) and [Type Relationships in LINQ Query Operations (C#)](http://msdn.microsoft.com/en-us/library/bb397924.aspx).

 Example

The following example shows a simple query over an [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx). Note that this example uses object initializers when the code calls the [Add](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.add.aspx) method, but this is not a requirement.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl02_code');" \o "Copy Code)

Imports System.Collections

Imports System.Linq

Module Module1

Public Class Student

Public Property FirstName As String

Public Property LastName As String

Public Property Scores As Integer()

End Class

Sub Main()

Dim student1 As New Student With {.FirstName = "Svetlana",

.LastName = "Omelchenko",

.Scores = New Integer() {98, 92, 81, 60}}

Dim student2 As New Student With {.FirstName = "Claire",

.LastName = "O'Donnell",

.Scores = New Integer() {75, 84, 91, 39}}

Dim student3 As New Student With {.FirstName = "Cesar",

.LastName = "Garcia",

.Scores = New Integer() {97, 89, 85, 82}}

Dim student4 As New Student With {.FirstName = "Sven",

.LastName = "Mortensen",

.Scores = New Integer() {88, 94, 65, 91}}

Dim arrList As New ArrayList()

arrList.Add(student1)

arrList.Add(student2)

arrList.Add(student3)

arrList.Add(student4)

' Use an explicit type for non-generic collections

Dim query = From student As Student In arrList

Where student.Scores(0) > 95

Select student

For Each student As Student In query

Console.WriteLine(student.LastName & ": " & student.Scores(0))

Next

' Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.")

Console.ReadKey()

End Sub

End Module

' Output:

' Omelchenko: 98

' Garcia: 97

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl03_code');" \o "Copy Code)

using System;

using System.Collections;

using System.Linq;

namespace NonGenericLINQ

{

public class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int[] Scores { get; set; }

}

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(

new Student

{

FirstName = "Svetlana", LastName = "Omelchenko", Scores = new int[] { 98, 92, 81, 60 }

});

arrList.Add(

new Student

{

FirstName = "Claire", LastName = "O’Donnell", Scores = new int[] { 75, 84, 91, 39 }

});

arrList.Add(

new Student

{

FirstName = "Sven", LastName = "Mortensen", Scores = new int[] { 88, 94, 65, 91 }

});

arrList.Add(

new Student

{

FirstName = "Cesar", LastName = "Garcia", Scores = new int[] { 97, 89, 85, 82 }

});

var query = from Student student in arrList

where student.Scores[0] > 95

select student;

foreach (Student s in query)

Console.WriteLine(s.LastName + ": " + s.Scores[0]);

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

}

/\* Output:

Omelchenko: 98

Garcia: 97

\*/

Visual Studio 2010

**How to: Query for Characters in a String (LINQ)**

Because the [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) class implements the generic [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface, any string can be queried as a sequence of characters. However, this is not a common use of LINQ. For complex pattern matching operations, use the [Regex](http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex.aspx) class.

 Example

The following example queries a string to determine the number of numeric digits it contains. Note that the query is "reused" after it is executed the first time. This is possible because the query itself does not store any actual results.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_ctl00_ctl00_code');" \o "Copy Code)

Class QueryAString

Shared Sub Main()

' A string is an IEnumerable data source.

Dim aString As String = "ABCDE99F-J74-12-89A"

' Select only those characters that are numbers

Dim stringQuery = From ch In aString

Where Char.IsDigit(ch)

Select ch

' Execute the query

For Each c As Char In stringQuery

Console.Write(c & " ")

Next

' Call the Count method on the existing query.

Dim count As Integer = stringQuery.Count()

Console.WriteLine(System.Environment.NewLine & "Count = " & count)

' Select all characters before the first '-'

Dim stringQuery2 = aString.TakeWhile(Function(c) c <> "-")

' Execute the second query

For Each ch In stringQuery2

Console.Write(ch)

Next

Console.WriteLine(System.Environment.NewLine & "Press any key to exit")

Console.ReadKey()

End Sub

End Class

' Output:

' 9 9 7 4 1 2 8 9

' Count = 8

' ABCDE99F

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_ctl00_ctl01_code');" \o "Copy Code)

class QueryAString

{

static void Main()

{

string aString = "ABCDE99F-J74-12-89A";

// Select only those characters that are numbers

IEnumerable<char> stringQuery =

from ch in aString

where Char.IsDigit(ch)

select ch;

// Execute the query

foreach (char c in stringQuery)

Console.Write(c + " ");

// Call the Count method on the existing query.

int count = stringQuery.Count();

Console.WriteLine("Count = {0}", count);

// Select all characters before the first '-'

IEnumerable<char> stringQuery2 = aString.TakeWhile(c => c != '-');

// Execute the second query

foreach (char c in stringQuery2)

Console.Write(c);

Console.WriteLine(System.Environment.NewLine + "Press any key to exit");

Console.ReadKey();

}

}

/\* Output:

Output: 9 9 7 4 1 2 8 9

Count = 8

ABCDE99F

\*/

Visual Studio 2010

**How to: Group Files by Extension (LINQ)**

This example shows how LINQ can be used to perform advanced grouping and sorting operations on lists of files or folders. It also shows how to page output in the console window by using the [Skip<(Of <(TSource>)>)](http://msdn.microsoft.com/en-us/library/bb358985.aspx) and [Take<(Of <(TSource>)>)](http://msdn.microsoft.com/en-us/library/bb503062.aspx) methods.

 Example

The following query shows how to group the contents of a specified directory tree by the file name extension.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl18_ctl00_ctl00_code');" \o "Copy Code)

Module GroupByExtension

Public Sub Main()

' Root folder to query, along with all subfolders.

Dim startFolder As String = "C:\program files\Microsoft Visual Studio 9.0\VB\"

' Used in WriteLine() to skip over startfolder in output lines.

Dim rootLength As Integer = startFolder.Length

'Take a snapshot of the folder contents

Dim dir As New System.IO.DirectoryInfo(startFolder)

Dim fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories)

' Create the query.

Dim queryGroupByExt = From file In fileList \_

Group By file.Extension.ToLower() Into fileGroup = Group \_

Order By ToLower \_

Select fileGroup

' Execute the query. By storing the result we can

' page the display with good performance.

Dim groupByExtList = queryGroupByExt.ToList()

' Display one group at a time. If the number of

' entries is greater than the number of lines

' in the console window, then page the output.

Dim trimLength = startFolder.Length

PageOutput(groupByExtList, trimLength)

End Sub

' Pages console diplay for large query results. No more than one group per page.

' This sub specifically works with group queries of FileInfo objects

' but can be modified for any type.

Sub PageOutput(ByVal groupQuery, ByVal charsToSkip)

' "3" = 1 line for extension key + 1 for "Press any key" + 1 for input cursor.

Dim numLines As Integer = Console.WindowHeight - 3

' Flag to indicate whether there are more results to diplay

Dim goAgain As Boolean = True

For Each fg As IEnumerable(Of System.IO.FileInfo) In groupQuery

' Start a new extension at the top of a page.

Dim currentLine As Integer = 0

Do While (currentLine < fg.Count())

Console.Clear()

Console.WriteLine(fg(0).Extension)

' Get the next page of results

' No more than one filename per page

Dim resultPage = From file In fg \_

Skip currentLine Take numLines

' Execute the query. Trim the display output.

For Each line In resultPage

Console.WriteLine(vbTab & line.FullName.Substring(charsToSkip))

Next

' Advance the current position

currentLine = numLines + currentLine

' Give the user a chance to break out of the loop

Console.WriteLine("Press any key for next page or the 'End' key to exit.")

Dim key As ConsoleKey = Console.ReadKey().Key

If key = ConsoleKey.End Then

goAgain = False

Exit For

End If

Loop

Next

End Sub

End Module

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl18_ctl00_ctl01_code');" \o "Copy Code)

class GroupByExtension

{

// This query will sort all the files under the specified folder

// and subfolder into groups keyed by the file extension.

private static void Main()

{

// Take a snapshot of the file system.

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\Common7";

// Used in WriteLine to trim output lines.

int trimLength = startFolder.Length;

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

// Create the query.

var queryGroupByExt =

from file in fileList

group file by file.Extension.ToLower() into fileGroup

orderby fileGroup.Key

select fileGroup;

// Display one group at a time. If the number of

// entries is greater than the number of lines

// in the console window, then page the output.

PageOutput(trimLength, queryGroupByExt);

}

// This method specifically handles group queries of FileInfo objects with string keys.

// It can be modified to work for any long listings of data. Note that explicit typing

// must be used in method signatures. The groupbyExtList parameter is a query that produces

// groups of FileInfo objects with string keys.

private static void PageOutput(int rootLength,

IEnumerable<System.Linq.IGrouping<string, System.IO.FileInfo>> groupByExtList)

{

// Flag to break out of paging loop.

bool goAgain = true;

// "3" = 1 line for extension + 1 for "Press any key" + 1 for input cursor.

int numLines = Console.WindowHeight - 3;

// Iterate through the outer collection of groups.

foreach (var filegroup in groupByExtList)

{

// Start a new extension at the top of a page.

int currentLine = 0;

// Output only as many lines of the current group as will fit in the window.

do

{

Console.Clear();

Console.WriteLine(filegroup.Key == String.Empty ? "[none]" : filegroup.Key);

// Get 'numLines' number of items starting at number 'currentLine'.

var resultPage = filegroup.Skip(currentLine).Take(numLines);

//Execute the resultPage query

foreach (var f in resultPage)

{

Console.WriteLine("\t{0}", f.FullName.Substring(rootLength));

}

// Increment the line counter.

currentLine += numLines;

// Give the user a chance to escape.

Console.WriteLine("Press any key to continue or the 'End' key to break...");

ConsoleKey key = Console.ReadKey().Key;

if (key == ConsoleKey.End)

{

goAgain = false;

break;

}

} while (currentLine < filegroup.Count());

if (goAgain == false)

break;

}

}

}

The output from this program can be long, depending on the details of the local file system and what the startFolder is set to. To enable viewing of all results, this example shows how to page through results. The same techniques can be applied to Windows and Web applications. Notice that because the code pages the items in a group, a nested **foreach** loop is required. There is also some additional logic to compute the current position in the list, and to enable the user to stop paging and exit the program. In this particular case, the paging query is run against the cached results from the original query. In other contexts, such as LINQ to SQL, such caching is not required.

Visual Studio 2010

**How to: Query An Assembly's Metadata with Reflection (LINQ)**

The following example shows how LINQ can be used with reflection to retrieve specific metadata about methods that match a specified search criterion. In this case, the query will find the names of all the methods in the assembly that return enumerable types such as arrays.

 Example

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl14_ctl00_ctl00_code');" \o "Copy Code)

Imports System.Reflection

Imports System.IO

Imports System.Linq

Module Module1

Sub Main()

Dim asmbly As Assembly =

Assembly.Load("System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken= b77a5c561934e089")

Dim pubTypesQuery = From type In asmbly.GetTypes()

Where type.IsPublic

From method In type.GetMethods()

Where method.ReturnType.IsArray = True

Let name = method.ToString()

Let typeName = type.ToString()

Group name By typeName Into methodNames = Group

Console.WriteLine("Getting ready to iterate")

For Each item In pubTypesQuery

Console.WriteLine(item.methodNames)

For Each type In item.methodNames

Console.WriteLine(" " & type)

Next

Next

Console.ReadKey()

End Sub

End Module

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl14_ctl00_ctl01_code');" \o "Copy Code)

using System.Reflection;

using System.IO;

namespace LINQReflection

{

class ReflectionHowTO

{

static void Main(string[] args)

{

Assembly assembly = Assembly.Load("System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken= b77a5c561934e089");

var pubTypesQuery = from type in assembly.GetTypes()

where type.IsPublic

from method in type.GetMethods()

where method.ReturnType.IsArray == true

|| ( method.ReturnType.GetInterface(

typeof(System.Collections.Generic.IEnumerable<>).FullName ) != null

&& method.ReturnType.FullName != "System.String" )

group method.ToString() by type.ToString();

foreach (var groupOfMethods in pubTypesQuery)

{

Console.WriteLine("Type: {0}", groupOfMethods.Key);

foreach (var method in groupOfMethods)

{

Console.WriteLine(" {0}", method);

}

}

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

}

}

The example uses the [GetTypes](http://msdn.microsoft.com/en-us/library/system.reflection.assembly.gettypes.aspx) method to return an array of types in the specified assembly. The [where](http://msdn.microsoft.com/en-us/library/bb311043.aspx) filter is applied so that only public types are returned. For each public type, a subquery is generated by using the [MethodInfo](http://msdn.microsoft.com/en-us/library/system.reflection.methodinfo.aspx) array that is returned from the [GetMethods](http://msdn.microsoft.com/en-us/library/system.type.getmethods.aspx) call. These results are filtered to return only those methods whose return type is an array or else a type that implements [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx). Finally, these results are grouped by using the type name as a key.

 Compiling the Code

* Create a Visual Studio project that targets the .NET Framework version 3.5. The project has a reference to System.Core.dll and a **using** directive (C#) or **Imports** statement (Visual Basic) for the System.Linq namespace by default. In C# projects, add a **using** directive for the System.IO namespace.
* Copy this code into your project.
* Press F5 to compile and run the program.
* Press any key to exit the console window.

Visual Studio 2010

**How to: Add Custom Methods for LINQ Queries**

You can extend the set of methods that you can use for LINQ queries by adding extension methods to the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface. For example, in addition to the standard average or maximum operations, you can create a custom aggregate method to compute a single value from a sequence of values. You can also create a method that works as a custom filter or a specific data transform for a sequence of values and returns a new sequence. Examples of such methods are [Distinct](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.distinct.aspx), [Skip<(Of <(TSource>)>)](http://msdn.microsoft.com/en-us/library/bb358985.aspx), and [Reverse<(Of <(TSource>)>)](http://msdn.microsoft.com/en-us/library/bb358497.aspx).

When you extend the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface, you can apply your custom methods to any enumerable collection. For more information, see [Extension Methods (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb383977.aspx) or [Extension Methods (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb384936.aspx).

 Adding an Aggregate Method

An aggregate method computes a single value from a set of values. LINQ provides several aggregate methods, including [Average](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.average.aspx), [Min](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.min.aspx), and [Max](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.max.aspx). You can create your own aggregate method by adding an extension method to the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface.

The following code example shows how to create an extension method called Median to compute a median for a sequence of numbers of type double.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl04_code');" \o "Copy Code)

Imports System.Runtime.CompilerServices

Module LINQExtension

' Extension method for the IEnumerable(of T) interface.

' The method accepts only values of the Double type.

<Extension()>

Function Median(ByVal source As IEnumerable(Of Double)) As Double

If source.Count = 0 Then

Throw New InvalidOperationException("Cannot compute median for an empty set.")

End If

Dim sortedSource = From number In source

Order By number

Dim itemIndex = sortedSource.Count \ 2

If sortedSource.Count Mod 2 = 0 Then

' Even number of items in list.

Return (sortedSource(itemIndex) + sortedSource(itemIndex - 1)) / 2

Else

' Odd number of items in list.

Return sortedSource(itemIndex)

End If

End Function

End Module

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl05_code');" \o "Copy Code)

public static class LINQExtension

{

public static double Median(this IEnumerable<double> source)

{

if (source.Count() == 0)

{

throw new InvalidOperationException("Cannot compute median for an empty set.");

}

var sortedList = from number in source

orderby number

select number;

int itemIndex = (int)sortedList.Count() / 2;

if (sortedList.Count() % 2 == 0)

{

// Even number of items.

return (sortedList.ElementAt(itemIndex) + sortedList.ElementAt(itemIndex - 1)) / 2;

}

else

{

// Odd number of items.

return sortedList.ElementAt(itemIndex);

}

}

}

You call this extension method for any enumerable collection in the same way you call other aggregate methods from the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface.

|  |
| --- |
| **NoteNote** |
| In Visual Basic, you can either use a method call or standard query syntax for the Aggregate or Group By clause. For more information, see [Aggregate Clause (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb531251.aspx) and [Group By Clause (Visual Basic)](http://msdn.microsoft.com/en-us/library/bb531412.aspx). |

The following code example shows how to use the Median method for an array of type double.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl10_code');" \o "Copy Code)

Dim numbers1() As Double = {1.9, 2, 8, 4, 5.7, 6, 7.2, 0}

Dim query1 = Aggregate num In numbers1 Into Median()

Console.WriteLine("Double: Median = " & query1)

...

' This code produces the following output:

'

' Double: Median = 4.85

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl11_code');" \o "Copy Code)

double[] numbers1 = { 1.9, 2, 8, 4, 5.7, 6, 7.2, 0 };

var query1 = numbers1.Median();

Console.WriteLine("double: Median = " + query1);

...

/\*

This code produces the following output:

Double: Median = 4.85

\*/

### Overloading an Aggregate Method to Accept Various Types

You can overload your aggregate method so that it accepts sequences of various types. The standard approach is to create an overload for each type. Another approach is to create an overload that will take a generic type and convert it to a specific type by using a delegate. You can also combine both approaches.

#### To create an overload for each type

You can create a specific overload for each type that you want to support. The following code example shows an overload of the Median method for the integer type.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl12_code');" \o "Copy Code)

' Integer overload

<Extension()>

Function Median(ByVal source As IEnumerable(Of Integer)) As Double

Return Aggregate num In source Select CDbl(num) Into med = Median()

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl13_code');" \o "Copy Code)

//int overload

public static double Median(this IEnumerable<int> source)

{

return (from num in source select (double)num).Median();

}

You can now call the Median overloads for both integer and double types, as shown in the following code:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl14_code');" \o "Copy Code)

Dim numbers1() As Double = {1.9, 2, 8, 4, 5.7, 6, 7.2, 0}

Dim query1 = Aggregate num In numbers1 Into Median()

Console.WriteLine("Double: Median = " & query1)

...

Dim numbers2() As Integer = {1, 2, 3, 4, 5}

Dim query2 = Aggregate num In numbers2 Into Median()

Console.WriteLine("Integer: Median = " & query2)

...

' This code produces the following output:

'

' Double: Median = 4.85

' Integer: Median = 3

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl15_code');" \o "Copy Code)

double[] numbers1 = { 1.9, 2, 8, 4, 5.7, 6, 7.2, 0 };

var query1 = numbers1.Median();

Console.WriteLine("double: Median = " + query1);

...

int[] numbers2 = { 1, 2, 3, 4, 5 };

var query2 = numbers2.Median();

Console.WriteLine("int: Median = " + query2);

...

/\*

This code produces the following output:

Double: Median = 4.85

Integer: Median = 3

\*/

#### To create a generic overload

You can also create an overload that accepts a sequence of generic objects. This overload takes a delegate as a parameter and uses it to convert a sequence of objects of a generic type to a specific type.

The following code shows an overload of the Median method that takes the [Func<(Of <(T, TResult>)>)](http://msdn.microsoft.com/en-us/library/bb549151.aspx) delegate as a parameter. This delegate takes an object of generic type T and returns an object of type double.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl17_code');" \o "Copy Code)

' Generic overload.

<Extension()>

Function Median(Of T)(ByVal source As IEnumerable(Of T),

ByVal selector As Func(Of T, Double)) As Double

Return Aggregate num In source Select selector(num) Into med = Median()

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl18_code');" \o "Copy Code)

// Generic overload.

public static double Median<T>(this IEnumerable<T> numbers,

Func<T, double> selector)

{

return (from num in numbers select selector(num)).Median();

}

You can now call the Median method for a sequence of objects of any type. If the type does not have its own method overload, you have to pass a delegate parameter. In Visual Basic and C#, you can use a lambda expression for this purpose. Also, in Visual Basic only, if you use the Aggregate or Group By clause instead of the method call, you can pass any value or expression that is in the scope this clause.

The following example code shows how to call the Median method for an array of integers and an array of strings. For strings, the median for the lengths of strings in the array is calculated. The example shows how to pass the [Func<(Of <(T, TResult>)>)](http://msdn.microsoft.com/en-us/library/bb549151.aspx) delegate parameter to the Median method for each case.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl20_code');" \o "Copy Code)

Dim numbers3() As Integer = {1, 2, 3, 4, 5}

' You can use num as a parameter for the Median method

' so that the compiler will implicitly convert its value to double.

' If there is no implicit conversion, the compiler will

' display an error message.

Dim query3 = Aggregate num In numbers3 Into Median(num)

Console.WriteLine("Integer: Median = " & query3)

Dim numbers4() As String = {"one", "two", "three", "four", "five"}

' With the generic overload, you can also use numeric properties of objects.

Dim query4 = Aggregate str In numbers4 Into Median(str.Length)

Console.WriteLine("String: Median = " & query4)

' This code produces the following output:

'

' Integer: Median = 3

' String: Median = 4

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl21_code');" \o "Copy Code)

int[] numbers3 = { 1, 2, 3, 4, 5 };

/\*

You can use the num=>num lambda expression as a parameter for the Median method

so that the compiler will implicitly convert its value to double.

If there is no implicit conversion, the compiler will display an error message.

\*/

var query3 = numbers3.Median(num => num);

Console.WriteLine("int: Median = " + query3);

string[] numbers4 = { "one", "two", "three", "four", "five" };

// With the generic overload, you can also use numeric properties of objects.

var query4 = numbers4.Median(str => str.Length);

Console.WriteLine("String: Median = " + query4);

/\*

This code produces the following output:

Integer: Median = 3

String: Median = 4

\*/

 Adding a Method That Returns a Collection

You can extend the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface with a custom query method that returns a sequence of values. In this case, the method must return a collection of type [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx). Such methods can be used to apply filters or data transforms to a sequence of values.

The following example shows how to create an extension method named AlternateElements that returns every other element in a collection, starting from the first element.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl02_code');" \o "Copy Code)

' Extension method for the IEnumerable(of T) interface.

' The method returns every other element of a sequence.

<Extension()>

Function AlternateElements(Of T)(

ByVal source As IEnumerable(Of T)

) As IEnumerable(Of T)

Dim list As New List(Of T)

Dim i = 0

For Each element In source

If (i Mod 2 = 0) Then

list.Add(element)

End If

i = i + 1

Next

Return list

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl03_code');" \o "Copy Code)

// Extension method for the IEnumerable<T> interface.

// The method returns every other element of a sequence.

public static IEnumerable<T> AlternateElements<T>(this IEnumerable<T> source)

{

List<T> list = new List<T>();

int i = 0;

foreach (var element in source)

{

if (i % 2 == 0)

{

list.Add(element);

}

i++;

}

return list;

}

You can call this extension method for any enumerable collection just as you would call other methods from the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface, as shown in the following code:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl05_code');" \o "Copy Code)

Dim strings() As String = {"a", "b", "c", "d", "e"}

Dim query = strings.AlternateElements()

For Each element In query

Console.WriteLine(element)

Next

' This code produces the following output:

'

' a

' c

' e

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl06_code');" \o "Copy Code)

string[] strings = { "a", "b", "c", "d", "e" };

var query = strings.AlternateElements();

foreach (var element in query)

{

Console.WriteLine(element);

}

/\*

This code produces the following output:

a

c

e

\*/