LINQ and Strings (Visual Basic)

Visual Studio 2015

LINQ can be used to query and transform strings and collections of strings. It can be especially useful with semi-structured data in text files. LINQ queries can be combined with traditional string functions and regular expressions. For example, you can use the Split or Split method to create an array of strings that you can then query or modify by using LINQ. You can use the IsMatch method in the **where** clause of a LINQ query. And you can use LINQ to query or modify the MatchCollection results returned by a regular expression.

You can also use the techniques described in this section to transform semi-structured text data to XML. For more information, see How to: Generate XML from CSV Files.

The examples in this section fall into two categories:

Querying a Block of Text

You can query, analyze, and modify text blocks by splitting them into a queryable array of smaller strings by using the Split method or the Split method. You can split the source text into words, sentences, paragraphs, pages, or any other criteria, and then perform additional splits if they are required in your query.

How to: Count Occurrences of a Word in a String (LINQ) (Visual Basic)
: Shows how to use LINQ for simple querying over text.

How to: Query for Sentences that Contain a Specified Set of Words (LINQ) (Visual Basic)
: Shows how to split text files on arbitrary boundaries and how to perform queries against each part.

How to: Query for Characters in a String (LINQ) (Visual Basic)
: Demonstrates that a string is a queryable type.

How to: Combine LINQ Queries with Regular Expressions (Visual Basic)
: Shows how to use regular expressions in LINQ queries for complex pattern matching on filtered query results.

Querying Semi-Structured Data in Text Format

Many different types of text files consist of a series of lines, often with similar formatting, such as tab- or comma-delimited files or fixed-length lines. After you read such a text file into memory, you can use LINQ to query and/or modify the lines. LINQ queries also simplify the task of combining data from multiple sources.

How to: Find the Set Difference Between Two Lists (LINQ) (Visual Basic)
: Shows how to find all the strings that are present in one list but not the other.

How to: Sort or Filter Text Data by Any Word or Field (LINQ) (Visual Basic)
: Shows how to sort text lines based on any word or field.

How to: Reorder the Fields of a Delimited File (LINQ) (Visual Basic)

Shows how to reorder fields in a line in a .csv file.

How to: Combine and Compare String Collections (LINQ) (Visual Basic)
Shows how to combine string lists in various ways.

How to: Populate Object Collections from Multiple Sources (LINQ) (Visual Basic)
Shows how to create object collections by using multiple text files as data sources.

How to: Join Content from Dissimilar Files (LINQ) (Visual Basic)
Shows how to combine strings in two lists into a single string by using a matching key.

How to: Split a File Into Many Files by Using Groups (LINQ) (Visual Basic)
Shows how to create new files by using a single file as a data source.

How to: Compute Column Values in a CSV Text File (LINQ) (Visual Basic)
Shows how to perform mathematical computations on text data in .csv files.

See Also

Language-Integrated Query (LINQ) (Visual Basic)
How to: Generate XML from CSV Files

How to: Count Occurrences of a Word in a String (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to use a LINQ query to count the occurrences of a specified word in a string. Note that to perform the count, first the Split method is called to create an array of words. There is a performance cost to the Split method. If the only operation on the string is to count the words, you should consider using the Matches or IndexOf methods instead. However, if performance is not a critical issue, or you have already split the sentence in order to perform other types of queries over it, then it makes sense to use LINQ to count the words or phrases as well.

Example

VB

```
Class CountWords

    Shared Sub Main()

        Dim text As String = "Historically, the world of data and the world of objects" &
                             " have not been well integrated. Programmers work in C# or Visual
Basic" &
                             " and also in SQL or XQuery. On the one side are concepts such as
classes," &
                             " objects, fields, inheritance, and .NET Framework APIs. On the other
side" &
                             " are tables, columns, rows, nodes, and separate languages for dealing
with" &
                             " them. Data types often require translation between the two worlds;
there are" &
                             " different standard functions. Because the object world has no notion
of query, a" &
                             " query can only be represented as a string without compile-time type
checking or" &
                             " IntelliSense support in the IDE. Transferring data from SQL tables or
XML trees to" &
                             " objects in memory is often tedious and error-prone."

        Dim searchTerm As String = "data"

        ' Convert the string into an array of words.
        Dim dataSource As String() = text.Split(New Char() {" ", ",", ".", ";", ":"},
                                                StringSplitOptions.RemoveEmptyEntries)

        ' Create and execute the query. It executes immediately
        ' because a singleton value is produced.
        ' Use ToLower to match "data" and "Data"
        Dim matchQuery = From word In dataSource
```

```
                        Where word.ToLowerInvariant() = searchTerm.ToLowerInvariant()
                        Select word

        ' Count the matches.
        Dim count As Integer = matchQuery.Count()
        Console.WriteLine(count & " occurrence(s) of the search term """ &
                          searchTerm & """ were found.")

        ' Keep console window open in debug mode.
        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()
    End Sub
End Class
' Output:
' 3 occurrence(s) of the search term "data" were found.
```

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)

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

Visual Studio 2015

Because the String class implements the generic IEnumerable(Of T) 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 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.

VB

```
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
```

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)
How to: Combine LINQ Queries with Regular Expressions (Visual Basic)

How to: Combine LINQ Queries with Regular Expressions (Visual Basic)

Visual Studio 2015

This example shows how to use the Regex class to create a regular expression for more complex matching in text strings. The LINQ query makes it easy to filter on exactly the files that you want to search with the regular expression, and to shape the results.

Example

VB

```
Class LinqRegExVB

    Shared Sub Main()

        ' Root folder to query, along with all subfolders.
        ' Modify this path as necessary so that it accesses your Visual Studio folder.
        Dim startFolder As String = "C:\program files\Microsoft Visual Studio 9.0\"
        ' One of the following paths may be more appropriate on your computer.
        'string startFolder = @"c:\program files (x86)\Microsoft Visual Studio 9.0\";
        'string startFolder = @"c:\program files\Microsoft Visual Studio 10.0\";
        'string startFolder = @"c:\program files (x86)\Microsoft Visual Studio 10.0\";

        ' Take a snapshot of the file system.
        Dim fileList As IEnumerable(Of System.IO.FileInfo) = GetFiles(startFolder)

        ' Create a regular expression to find all things "Visual".
        Dim searchTerm As System.Text.RegularExpressions.Regex =
            New System.Text.RegularExpressions.Regex("Visual (Basic|C#|C\+
\+|J#|SourceSafe|Studio)")

        ' Search the contents of each .htm file.
        ' Remove the where clause to find even more matches!
        ' This query produces a list of files where a match
        ' was found, and a list of the matches in that file.
        ' Note: Explicit typing of "Match" in select clause.
        ' This is required because MatchCollection is not a
        ' generic IEnumerable collection.
        Dim queryMatchingFiles = From afile In fileList
                                 Where afile.Extension = ".htm"
                                 Let fileText = System.IO.File.ReadAllText(afile.FullName)
                                 Let matches = searchTerm.Matches(fileText)
                                 Where (matches.Count > 0)
                                 Select Name = afile.FullName,
                                        Matches = From match As
System.Text.RegularExpressions.Match In matches
```

```
                                                Select match.Value

        ' Execute the query.
        Console.WriteLine("The term " & searchTerm.ToString() & " was found in:")

        For Each fileMatches In queryMatchingFiles
            ' Trim the path a bit, then write
            ' the file name in which a match was found.
            Dim s = fileMatches.Name.Substring(startFolder.Length - 1)
            Console.WriteLine(s)

            ' For this file, write out all the matching strings
            For Each match In fileMatches.Matches
                Console.WriteLine("  " + match)
            Next
        Next

        ' Keep the console window open in debug mode
        Console.WriteLine("Press any key to exit")
        Console.ReadKey()
    End Sub

    ' Function to retrieve a list of files. Note that this is a copy
    ' of the file information.
    Shared Function GetFiles(ByVal root As String) As IEnumerable(Of System.IO.FileInfo)
        Return From file In My.Computer.FileSystem.GetFiles(
                   root, FileIO.SearchOption.SearchAllSubDirectories, "*.*")
               Select New System.IO.FileInfo(file)
    End Function

End Class
```

Note that you can also query the MatchCollection object that is returned by a **RegEx** search. In this example only the value of each match is produced in the results. However, it is also possible to use LINQ to perform all kinds of filtering, sorting, and grouping on that collection. Because MatchCollection is a non-generic IEnumerable collection, you have to explicitly state the type of the range variable in the query.

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)
LINQ and File Directories (Visual Basic)

How to: Find the Set Difference Between Two Lists (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to use LINQ to compare two lists of strings and output those lines that are in names1.txt but not in names2.txt.

To create the data files

1. Copy names1.txt and names2.txt to your solution folder as shown in How to: Combine and Compare String Collections (LINQ) (Visual Basic).

Example

VB

```
Class CompareLists

    Shared Sub Main()

        ' Create the IEnumerable data sources.
        Dim names1 As String() = System.IO.File.ReadAllLines("../../../names1.txt")
        Dim names2 As String() = System.IO.File.ReadAllLines("../../../names2.txt")

        ' Create the query. Note that method syntax must be used here.
        Dim differenceQuery = names1.Except(names2)
        Console.WriteLine("The following lines are in names1.txt but not names2.txt")

        ' Execute the query.
        For Each name As String In differenceQuery
            Console.WriteLine(name)
        Next

        ' Keep console window open in debug mode.
        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()
    End Sub
End Class
' Output:
' The following lines are in names1.txt but not names2.txt
' Potra, Cristina
' Noriega, Fabricio
' Aw, Kam Foo
' Toyoshima, Tim
' Guy, Wey Yuan
```

```
    ' Garcia, Debra
```

Some types of query operations in Visual Basic, such as Except(Of TSource), Distinct(Of TSource), Union(Of TSource), and Concat(Of TSource), can only be expressed in method-based syntax.

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)

How to: Find the Set Difference Between Two Lists (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to use LINQ to compare two lists of strings and output those lines that are in names1.txt but not in names2.txt.

To create the data files

1. Copy names1.txt and names2.txt to your solution folder as shown in How to: Combine and Compare String Collections (LINQ) (Visual Basic).

Example

VB

```
Class CompareLists

    Shared Sub Main()

        ' Create the IEnumerable data sources.
        Dim names1 As String() = System.IO.File.ReadAllLines("../../../names1.txt")
        Dim names2 As String() = System.IO.File.ReadAllLines("../../../names2.txt")

        ' Create the query. Note that method syntax must be used here.
        Dim differenceQuery = names1.Except(names2)
        Console.WriteLine("The following lines are in names1.txt but not names2.txt")

        ' Execute the query.
        For Each name As String In differenceQuery
            Console.WriteLine(name)
        Next

        ' Keep console window open in debug mode.
        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()
    End Sub
End Class
' Output:
' The following lines are in names1.txt but not names2.txt
' Potra, Cristina
' Noriega, Fabricio
' Aw, Kam Foo
' Toyoshima, Tim
' Guy, Wey Yuan
```

```
    ' Garcia, Debra
```

Some types of query operations in Visual Basic, such as Except(Of TSource), Distinct(Of TSource), Union(Of TSource), and Concat(Of TSource), can only be expressed in method-based syntax.

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)

How to: Reorder the Fields of a Delimited File (LINQ) (Visual Basic)

Visual Studio 2015

A comma-separated value (CSV) file is a text file that is often used to store spreadsheet data or other tabular data that is represented by rows and columns. By using the Split method to separate the fields, it is very easy to query and manipulate CSV files by using LINQ. In fact, the same technique can be used to reorder the parts of any structured line of text; it is not limited to CSV files.

In the following example, assume that the three columns represent students' "last name," "first name", and "ID." The fields are in alphabetical order based on the students' last names. The query produces a new sequence in which the ID column appears first, followed by a second column that combines the student's first name and last name. The lines are reordered according to the ID field. The results are saved into a new file and the original data is not modified.

To create the data file

1. Copy the following lines into a plain text file that is named spreadsheet1.csv. Save the file in your project folder.

```
Adams,Terry,120
Fakhouri,Fadi,116
Feng,Hanying,117
Garcia,Cesar,114
Garcia,Debra,115
Garcia,Hugo,118
Mortensen,Sven,113
O'Donnell,Claire,112
Omelchenko,Svetlana,111
Tucker,Lance,119
Tucker,Michael,122
Zabokritski,Eugene,121
```

Example

VB

```
Class CSVFiles

    Shared Sub Main()

        ' Create the IEnumerable data source.
        Dim lines As String() = System.IO.File.ReadAllLines("../../../spreadsheet1.csv")

        ' Execute the query. Put field 2 first, then
```

```
        ' reverse and combine fields 0 and 1 from the old field
        Dim lineQuery = From line In lines
                        Let x = line.Split(New Char() {","})
                        Order By x(2)
                        Select x(2) & ", " & (x(1) & " " & x(0))

        ' Execute the query and write out the new file. Note that WriteAllLines
        ' takes a string array, so ToArray is called on the query.
        System.IO.File.WriteAllLines("../../../spreadsheet2.csv", lineQuery.ToArray())

        ' Keep console window open in debug mode.
        Console.WriteLine("Spreadsheet2.csv written to disk. Press any key to exit")
        Console.ReadKey()
    End Sub
End Class
' Output to spreadsheet2.csv:
' 111, Svetlana Omelchenko
' 112, Claire O'Donnell
' 113, Sven Mortensen
' 114, Cesar Garcia
' 115, Debra Garcia
' 116, Fadi Fakhouri
' 117, Hanying Feng
' 118, Hugo Garcia
' 119, Lance Tucker
' 120, Terry Adams
' 121, Eugene Zabokritski
' 122, Michael Tucker
```

Compiling the Code

See Also

LINQ and Strings (Visual Basic)
LINQ and File Directories (Visual Basic)
How to: Generate XML from CSV Files

How to: Combine and Compare String Collections (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to merge files that contain lines of text and then sort the results. Specifically, it shows how to perform a simple concatenation, a union, and an intersection on the two sets of text lines.

To set up the project and the text files

1. Copy these names into a text file that is named names1.txt and save it in your project folder:

```
Bankov, Peter
Holm, Michael
Garcia, Hugo
Potra, Cristina
Noriega, Fabricio
Aw, Kam Foo
Beebe, Ann
Toyoshima, Tim
Guy, Wey Yuan
Garcia, Debra
```

2. Copy these names into a text file that is named names2.txt and save it in your project folder. Note that the two files have some names in common.

```
Liu, Jinghao
Bankov, Peter
Holm, Michael
Garcia, Hugo
Beebe, Ann
Gilchrist, Beth
Myrcha, Jacek
Giakoumakis, Leo
McLin, Nkenge
El Yassir, Mehdi
```

Example

VB

```
Class ConcatenateStrings
```

```
    Shared Sub Main()

        ' Create the IEnumerable data sources.
        Dim fileA As String() = System.IO.File.ReadAllLines("../../../names1.txt")
        Dim fileB As String() = System.IO.File.ReadAllLines("../../../names2.txt")

        ' Simple concatenation and sort.
        Dim concatQuery = fileA.Concat(fileB).OrderBy(Function(name) name)

        ' Pass the query variable to another function for execution
        OutputQueryResults(concatQuery, "Simple concatenation and sort. Duplicates are
preserved:")

        ' New query. Concatenate files and remove duplicates
        Dim uniqueNamesQuery = fileA.Union(fileB).OrderBy(Function(name) name)
        OutputQueryResults(uniqueNamesQuery, "Union removes duplicate names:")

        ' New query. Find the names that occur in both files.
        Dim commonNamesQuery = fileA.Intersect(fileB)
        OutputQueryResults(commonNamesQuery, "Merge based on intersect: ")

        ' New query in three steps for better readability
        ' First filter each list separately

        Dim nameToSearch As String = "Garcia"
        Dim mergeQueryA As IEnumerable(Of String) = From name In fileA
                          Let n = name.Split(New Char() {","})
                          Where n(0) = nameToSearch
                          Select name

        Dim mergeQueryB = From name In fileB
                          Let n = name.Split(New Char() {","})
                          Where n(0) = nameToSearch
                          Select name

        ' Create a new query to concatenate and sort results. Duplicates are removed in
Union.
        ' Note that none of the queries actually executed until the call to
OutputQueryResults.
        Dim mergeSortQuery = mergeQueryA.Union(mergeQueryB).OrderBy(Function(str) str)

        ' Now execute mergeSortQuery
        OutputQueryResults(mergeSortQuery, "Concat based on partial name match """ &
nameToSearch & """ from each list:")

        ' Keep console window open in debug mode.
        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()

    End Sub

    Shared Sub OutputQueryResults(ByVal query As IEnumerable(Of String), ByVal message As
String)
```

```
        Console.WriteLine(System.Environment.NewLine & message)
        For Each item As String In query
            Console.WriteLine(item)
        Next
        Console.WriteLine(query.Count & " total names in list")

    End Sub
End Class
' Output:

' Simple concatenation and sort. Duplicates are preserved:
' Aw, Kam Foo
' Bankov, Peter
' Bankov, Peter
' Beebe, Ann
' Beebe, Ann
' El Yassir, Mehdi
' Garcia, Debra
' Garcia, Hugo
' Garcia, Hugo
' Giakoumakis, Leo
' Gilchrist, Beth
' Guy, Wey Yuan
' Holm, Michael
' Holm, Michael
' Liu, Jinghao
' McLin, Nkenge
' Myrcha, Jacek
' Noriega, Fabricio
' Potra, Cristina
' Toyoshima, Tim
' 20 total names in list

' Union removes duplicate names:
' Aw, Kam Foo
' Bankov, Peter
' Beebe, Ann
' El Yassir, Mehdi
' Garcia, Debra
' Garcia, Hugo
' Giakoumakis, Leo
' Gilchrist, Beth
' Guy, Wey Yuan
' Holm, Michael
' Liu, Jinghao
' McLin, Nkenge
' Myrcha, Jacek
' Noriega, Fabricio
' Potra, Cristina
' Toyoshima, Tim
' 16 total names in list

' Merge based on intersect:
' Bankov, Peter
```

```
' Holm, Michael
' Garcia, Hugo
' Beebe, Ann
' 4 total names in list

' Concat based on partial name match "Garcia" from each list:
' Garcia, Debra
' Garcia, Hugo
' 2 total names in list
```

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)
LINQ and File Directories (Visual Basic)

How to: Populate Object Collections from Multiple Sources (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to merge data from different sources into a sequence of new types.

Note

Do not try to join in-memory data or data in the file system with data that is still in a database. Such cross-domain joins can yield undefined results because of different ways in which join operations might be defined for database queries and other types of sources. Additionally, there is a risk that such an operation could cause an out-of-memory exception if the amount of data in the database is large enough. To join data from a database to in-memory data, first call **ToList** or **ToArray** on the database query, and then perform the join on the returned collection.

To create the data file

- Copy the names.csv and scores.csv files into your project folder, as described in How to: Join Content from Dissimilar Files (LINQ) (Visual Basic).

Example

The following example shows how to use a named type `Student` to store merged data from two in-memory collections of strings that simulate spreadsheet data in .csv format. The first collection of strings represents the student names and IDs, and the second collection represents the student ID (in the first column) and four exam scores. The ID is used as the foreign key.

VB

```
Class Student
    Public FirstName As String
    Public LastName As String
    Public ID As Integer
    Public ExamScores As List(Of Integer)
End Class

Class PopulateCollection

    Shared Sub Main()

        ' Merge content from spreadsheets into a list of Student objects.

        ' These data files are defined in How to: Join Content from
        ' Dissimilar Files (LINQ).
```

```
        ' Each line of names.csv consists of a last name, a first name, and an
        ' ID number, separated by commas. For example, Omelchenko,Svetlana,111
        Dim names As String() = System.IO.File.ReadAllLines("../../../names.csv")

        ' Each line of scores.csv consists of an ID number and four test
        ' scores, separated by commas. For example, 111, 97, 92, 81, 60
        Dim scores As String() = System.IO.File.ReadAllLines("../../../scores.csv")

        ' The following query merges the content of two dissimilar spreadsheets
        ' based on common ID values.
        ' Multiple From clauses are used instead of a Join clause
        ' in order to store the results of scoreLine.Split.
        ' Note the dynamic creation of a list of integers for the
        ' ExamScores member. We skip the first item in the split string
        ' because it is the student ID, not an exam score.
        Dim queryNamesScores = From nameLine In names
                               Let splitName = nameLine.Split(New Char() {","})
                               From scoreLine In scores
                               Let splitScoreLine = scoreLine.Split(New Char() {","})
                               Where splitName(2) = splitScoreLine(0)
                               Select New Student() With {
                                    .FirstName = splitName(0), .LastName = splitName(1), .ID =
splitName(2),
                                    .ExamScores = (From scoreAsText In splitScoreLine Skip 1
                                                   Select
Convert.ToInt32(scoreAsText)).ToList()}

        ' Optional. Store the query results for faster access in future
        ' queries. This could be useful with very large data files.
        Dim students As List(Of Student) = queryNamesScores.ToList()

        ' Display each student's name and exam score average.
        For Each s In students
            Console.WriteLine("The average score of " & s.FirstName & " " &
                              s.LastName & " is " & s.ExamScores.Average())
        Next

        ' Keep console window open in debug mode.
        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()
    End Sub
End Class

' Output:
' The average score of Omelchenko Svetlana is 82.5
' The average score of O'Donnell Claire is 72.25
' The average score of Mortensen Sven is 84.5
' The average score of Garcia Cesar is 88.25
' The average score of Garcia Debra is 67
' The average score of Fakhouri Fadi is 92.25
' The average score of Feng Hanying is 88
' The average score of Garcia Hugo is 85.75
' The average score of Tucker Lance is 81.75
```

```
' The average score of Adams Terry is 85.25
' The average score of Zabokritski Eugene is 83
' The average score of Tucker Michael is 92
```

In the Select Clause (Visual Basic) clause, an object initializer is used to instantiate each new `Student` object by using the data from the two sources.

If you do not have to store the results of a query, anonymous types can be more convenient than named types. Named types are required if you pass the query results outside the method in which the query is executed. The following example performs the same task as the previous example, but uses anonymous types instead of named types:

VB

```
' Merge the data by using an anonymous type.
' Note the dynamic creation of a list of integers for the
' ExamScores member. We skip 1 because the first string
' in the array is the student ID, not an exam score.
Dim queryNamesScores2 =
    From nameLine In names
    Let splitName = nameLine.Split(New Char() {","})
    From scoreLine In scores
    Let splitScoreLine = scoreLine.Split(New Char() {","})
    Where splitName(2) = splitScoreLine(0)
    Select New With
           {.Last = splitName(0),
            .First = splitName(1),
            .ExamScores = (From scoreAsText In splitScoreLine Skip 1
                           Select Convert.ToInt32(scoreAsText)).ToList()}

' Display each student's name and exam score average.
For Each s In queryNamesScores2
    Console.WriteLine("The average score of " & s.First & " " &
                      s.Last & " is " & s.ExamScores.Average())
Next
```

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)

How to: Join Content from Dissimilar Files (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to join data from two comma-delimited files that share a common value that is used as a matching key. This technique can be useful if you have to combine data from two spreadsheets, or from a spreadsheet and from a file that has another format, into a new file. You can modify the example to work with any kind of structured text.

To create the data files

1. Copy the following lines into a file that is named scores.csv and save it to your project folder. The file represents spreadsheet data. Column 1 is the student's ID, and columns 2 through 5 are test scores.

```
111, 97, 92, 81, 60
112, 75, 84, 91, 39
113, 88, 94, 65, 91
114, 97, 89, 85, 82
115, 35, 72, 91, 70
116, 99, 86, 90, 94
117, 93, 92, 80, 87
118, 92, 90, 83, 78
119, 68, 79, 88, 92
120, 99, 82, 81, 79
121, 96, 85, 91, 60
122, 94, 92, 91, 91
```

2. Copy the following lines into a file that is named names.csv and save it to your project folder. The file represents a spreadsheet that contains the student's last name, first name, and student ID.

```
Omelchenko,Svetlana,111
O'Donnell,Claire,112
Mortensen,Sven,113
Garcia,Cesar,114
Garcia,Debra,115
Fakhouri,Fadi,116
Feng,Hanying,117
Garcia,Hugo,118
Tucker,Lance,119
Adams,Terry,120
Zabokritski,Eugene,121
Tucker,Michael,122
```

Example

VB

```
Class JoinStrings

    Shared Sub Main()

        ' Join content from spreadsheet files that contain
        ' related information. names.csv contains the student name
        ' plus an ID number. scores.csv contains the ID and a
        ' set of four test scores. The following query joins
        ' the scores to the student names by using ID as a
        ' matching key.

        Dim names As String() = System.IO.File.ReadAllLines("../../../names.csv")
        Dim scores As String() = System.IO.File.ReadAllLines("../../../scores.csv")

        ' Name:    Last[0],       First[1],  ID[2],     Grade Level[3]
        '          Omelchenko,    Svetlana,  111,       2
        ' Score:   StudentID[0],  Exam1[1]   Exam2[2],  Exam3[3],  Exam4[4]
        '          111,           97,        92,        81,        60

        ' This query joins two dissimilar spreadsheets based on common ID value.
        ' Multiple from clauses are used instead of a join clause
        ' in order to store results of id.Split.
        Dim scoreQuery1 = From name In names
                          Let n = name.Split(New Char() {","})
                             From id In scores
                             Let n2 = id.Split(New Char() {","})
                             Where n(2) = n2(0)
                             Select n(0) & "," & n(1) & "," & n2(0) & "," & n2(1) & "," &
                               n2(2) & "," & n2(3)

        ' Pass a query variable to a Sub and execute it there.
        ' The query itself is unchanged.
        OutputQueryResults(scoreQuery1, "Merge two spreadsheets:")

        ' Keep console window open in debug mode.
        Console.WriteLine("Press any key to exit.")
        Console.ReadKey()
    End Sub

    Shared Sub OutputQueryResults(ByVal query As IEnumerable(Of String), ByVal message As
String)

        Console.WriteLine(System.Environment.NewLine & message)
        For Each item As String In query
            Console.WriteLine(item)
        Next
        Console.WriteLine(query.Count & " total names in list")

    End Sub
```

```
End Class
' Output:
'Merge two spreadsheets:
'Adams,Terry,120, 99, 82, 81
'Fakhouri,Fadi,116, 99, 86, 90
'Feng,Hanying,117, 93, 92, 80
'Garcia,Cesar,114, 97, 89, 85
'Garcia,Debra,115, 35, 72, 91
'Garcia,Hugo,118, 92, 90, 83
'Mortensen,Sven,113, 88, 94, 65
'O'Donnell,Claire,112, 75, 84, 91
'Omelchenko,Svetlana,111, 97, 92, 81
'Tucker,Lance,119, 68, 79, 88
'Tucker,Michael,122, 94, 92, 91
'Zabokritski,Eugene,121, 96, 85, 91
'12 total names in list
```

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)
LINQ and File Directories (Visual Basic)

How to: Split a File Into Many Files by Using Groups (LINQ) (Visual Basic)

Visual Studio 2015

This example shows one way to merge the contents of two files and then create a set of new files that organize the data in a new way.

To create the data files

1. Copy these names into a text file that is named names1.txt and save it in your project folder:

```
Bankov, Peter
Holm, Michael
Garcia, Hugo
Potra, Cristina
Noriega, Fabricio
Aw, Kam Foo
Beebe, Ann
Toyoshima, Tim
Guy, Wey Yuan
Garcia, Debra
```

2. Copy these names into a text file that is named names2.txt and save it in your project folder: Note that the two files have some names in common.

```
Liu, Jinghao
Bankov, Peter
Holm, Michael
Garcia, Hugo
Beebe, Ann
Gilchrist, Beth
Myrcha, Jacek
Giakoumakis, Leo
McLin, Nkenge
El Yassir, Mehdi
```

Example

VB

```
Class SplitWithGroups
```

```
    Shared Sub Main()

        Dim fileA As String() = System.IO.File.ReadAllLines("../../../names1.txt")
        Dim fileB As String() = System.IO.File.ReadAllLines("../../../names2.txt")

        ' Concatenate and remove duplicate names based on
        Dim mergeQuery As IEnumerable(Of String) = fileA.Union(fileB)

        ' Group the names by the first letter in the last name
        Dim groupQuery = From name In mergeQuery
                         Let n = name.Split(New Char() {","})
                         Order By n(0)
                         Group By groupKey = n(0)(0)
                         Into groupName = Group

        ' Create a new file for each group that was created
        ' Note that nested foreach loops are required to access
        ' individual items with each group.
        For Each gGroup In groupQuery
            Dim fileName As String = "..'..'..'testFile_" & gGroup.groupKey & ".txt"
            Dim sw As New System.IO.StreamWriter(fileName)
            Console.WriteLine(gGroup.groupKey)
            For Each item In gGroup.groupName
                Console.WriteLine("   " & item.name)
                sw.WriteLine(item.name)
            Next
            sw.Close()
        Next

        ' Keep console window open in debug mode.
        Console.WriteLine("Files have been written. Press any key to exit.")
        Console.ReadKey()

    End Sub
End Class
' Console Output:
' A
'    Aw, Kam Foo
' B
'    Bankov, Peter
'    Beebe, Ann
' E
'    El Yassir, Mehdi
' G
'    Garcia, Hugo
'    Garcia, Debra
'    Giakoumakis, Leo
'    Gilchrist, Beth
'    Guy, Wey Yuan
' H
'    Holm, Michael
' L
'    Liu, Jinghao
```

```
    ' M
    '    McLin, Nkenge
    '    Myrcha, Jacek
    ' N
    '    Noriega, Fabricio
    ' P
    '    Potra, Cristina
    ' T
    '    Toyoshima, Tim
```

The program writes a separate file for each group in the same folder as the data files.

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)
LINQ and File Directories (Visual Basic)

How to: Compute Column Values in a CSV Text File (LINQ) (Visual Basic)

Visual Studio 2015

This example shows how to perform aggregate computations such as Sum, Average, Min, and Max on the columns of a .csv file. The example principles that are shown here can be applied to other types of structured text.

To create the source file

1. Copy the following lines into a file that is named scores.csv and save it in your project folder. Assume that the first column represents a student ID, and subsequent columns represent scores from four exams.

```
111, 97, 92, 81, 60
112, 75, 84, 91, 39
113, 88, 94, 65, 91
114, 97, 89, 85, 82
115, 35, 72, 91, 70
116, 99, 86, 90, 94
117, 93, 92, 80, 87
118, 92, 90, 83, 78
119, 68, 79, 88, 92
120, 99, 82, 81, 79
121, 96, 85, 91, 60
122, 94, 92, 91, 91
```

Example

VB

```
Class SumColumns

    Public Shared Sub Main()

        Dim lines As String() = System.IO.File.ReadAllLines("../../../scores.csv")

        ' Specifies the column to compute
        ' This value could be passed in at runtime.
        Dim exam = 3

        ' Spreadsheet format:
        ' Student ID    Exam#1  Exam#2  Exam#3  Exam#4
        ' 111,          97,     92,     81,     60
        ' one is added to skip over the first column
```

```
        ' which holds the student ID.
        SumColumn(lines, exam + 1)
        Console.WriteLine()
        MultiColumns(lines)

        ' Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit...")
        Console.ReadKey()

    End Sub

    Shared Sub SumColumn(ByVal lines As IEnumerable(Of String), ByVal col As Integer)

        ' This query performs two steps:
        ' split the string into a string array
        ' convert the specified element to
        ' integer and select it.
        Dim columnQuery = From line In lines
                          Let x = line.Split(",")
                          Select Convert.ToInt32(x(col))

        ' Execute and cache the results for performance.
        ' Only needed with very large files.
        Dim results = columnQuery.ToList()

        ' Perform aggregate calculations
        ' on the column specified by col.
        Dim avgScore = Aggregate score In results Into Average(score)
        Dim minScore = Aggregate score In results Into Min(score)
        Dim maxScore = Aggregate score In results Into Max(score)

        Console.WriteLine("Single Column Query:")
        Console.WriteLine("Exam #{0}: Average:{1:##.##} High Score:{2} Low Score:{3}",
                    col, avgScore, maxScore, minScore)


    End Sub

    Shared Sub MultiColumns(ByVal lines As IEnumerable(Of String))

        Console.WriteLine("Multi Column Query:")

        ' Create the query. It will produce nested sequences.
        ' multiColQuery performs these steps:
        ' 1) convert the string to a string array
        ' 2) skip over the "Student ID" column and take the rest
        ' 3) convert each field to an int and select that
        '    entire sequence as one row in the results.
        Dim multiColQuery = From line In lines
                            Let fields = line.Split(",")
                            Select From str In fields Skip 1
                                        Select Convert.ToInt32(str)

        Dim results = multiColQuery.ToList()
```

```
        ' Find out how many columns we have.
        Dim columnCount = results(0).Count()

        ' Perform aggregate calculations on each column.
        ' One loop for each score column in scores.
        ' We can use a for loop because we have already
        ' executed the multiColQuery in the call to ToList.

        For j As Integer = 0 To columnCount - 1
            Dim column = j
            Dim res2 = From row In results
                       Select row.ElementAt(column)

            ' Perform aggregate calculations
            ' on the column specified by col.
            Dim avgScore = Aggregate score In res2 Into Average(score)
            Dim minScore = Aggregate score In res2 Into Min(score)
            Dim maxScore = Aggregate score In res2 Into Max(score)

            ' Add 1 to column numbers because exams in this course start with #1
            Console.WriteLine("Exam #{0} Average: {1:##.##} High Score: {2} Low Score:
{3}",
                              column + 1, avgScore, maxScore, minScore)

        Next
    End Sub

End Class
' Output:
' Single Column Query:
' Exam #4: Average:76.92 High Score:94 Low Score:39

' Multi Column Query:
' Exam #1 Average: 86.08 High Score: 99 Low Score: 35
' Exam #2 Average: 86.42 High Score: 94 Low Score: 72
' Exam #3 Average: 84.75 High Score: 91 Low Score: 65
' Exam #4 Average: 76.92 High Score: 94 Low Score: 39
```

The query works by using the Split method to convert each line of text into an array. Each array element represents a column. Finally, the text in each column is converted to its numeric representation. If your file is a tab-separated file, just update the argument in the **Split** method to `\t`.

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher with a reference to System.Core.dll and a **Imports** statement for the System.Linq namespace.

See Also

LINQ and Strings (Visual Basic)

LINQ and File Directories (Visual Basic)