9

Introduction to LINQ and Generic Collections

To write it, it took three months;
to conceive it three minutes;
to collect the data in it—all my life.

– **F. Scott Fitzgerald**

Science is feasible when the variables are few and can be enumerated; when their combinations are distinct and clear.

– **Paul Valéry**

You shall listen to all sides and filter them from your self.

– **Walt Whitman**

The portraitist can select one tiny aspect of everything shown at a moment to incorporate into the final painting.

– **Robert Nozick**

List, list, O, list!

– **William Shakespeare**

Be wise to-day; 't is madness to defer.

– **Edward Young**

OBJECTIVES

In this chapter you will learn:

- Basic LINQ concepts.
- How to query an array using LINQ.
- Basic .NET collections concepts.
- How to create and use a generic `List` collection.
- How to write a generic method.
- How to query a generic `List` collection using LINQ.

Outline

9.1 Introduction

- Collections offer greater capabilities than arrays.
- `Lists` automatically change their size.
- Large amounts of data are often stored in a database.
- A database management system (DBMS) is used to control data.

9.1 Introduction (Cont.)

- SQL is the international standard used with relational databases.
- **LINQ** (**Language-Integrated Query**) allows you to write **query expressions** to retrieve information.
- **LINQ to Objects** can query arrays and `Lists`, selecting elements that satisfy a set of conditions.

9.1 Introduction (Cont.)

Chapter	Used to
Chapter 9, Introduction to LINQ and Generic Collections	Query arrays and Lists.
Chapter 17, Graphics and Multimedia	Select GUI controls in a Windows Forms application.
Chapter 18, Files and Streams	Search a directory and manipulate text files.
Chapter 19, XML and LINQ to XML	Query an XML document.
Chapter 20, Databases and LINQ to SQL	Retrieve information from a database; insert data in a database.
Chapter 21, ASP.NET and ASP.NET Ajax	Retrieve information from a database to be used in a web-based application.
Chapter 22, Windows Communication Foundation (WCF) Web Services	Query and update a database. Process XML returned by WCF services.
Chapter 23, Silverlight, Rich Internet Applications and Multimedia	Process XML returned by web services to a Silverlight application.
Chapter 24, Data Structures and Generic Collections	Query .NET collections.

Fig. 9.1 | LINQ usage throughout the book.

Outline

- Figure 9.2 demonstrates querying an array using LINQ.

LINQWithSimpleTypeArray.vb

(1 of 4)

```
1  ' Fig. 9.2: LINQWithSimpleTypeArray.vb
2  ' LINQ to Objects using an Integer array.
3  Module LINQWithSimpleTypeArray
4     Sub Main()
5        ' create an integer array
6        Dim values As Integer() = {2, 9, 5, 0, 3, 7, 1, 4, 8, 5}
7
8        Display(values, "Original array:") ' display original values
9
10       ' LINQ query that obtains values greater than 4 from the array
11       Dim filtered = _
12          From value In values _
13          Where value > 4 _
14          Select value
15
```

Returns all `Integer`s in the array greater than `4`.

Fig. 9.2 | LINQ to Objects using an `Integer` array. (Part 1 of 4.)

Outline

```
16      ' display filtered results
17      Display(filtered, "Array values greater than 4:")
18
19      ' use Order By clause to sort original array in ascending order
20      Dim sorted = _
21         From value In values _
22         Order By value _
23         Select value
24
25      Display(sorted, "Original array, sorted:") ' display sorted results
26
27      ' sort the filtered results into descending order
28      Dim sortFilteredResults = _
29         From value In filtered _
30         Order By value Descending _
31         Select value
32
33      ' display the sorted results
34      Display(sortFilteredResults, _
35         "Values greater than 4, descending order (separately):")
```

LINQWithSimpleTypeArray.vb

(2 of 4)

`Order By` sorts results in ascending order.

The `Descending` query operator sorts results in descending order.

Fig. 9.2 | LINQ to Objects using an `Integer` array. (Part 2 of 4.)

Outline

```
36
37       ' filter original array and sort in descending order
38       Dim sortAndFilter = _
39          From value In values _
40          Where value > 4 _
41          Order By value Descending _
42          Select value
43
44       ' display the filtered and sorted results
45       Display(sortAndFilter, _
46          "Values greater than 4, descending order (one query):")
47    End Sub ' Main
48
49    ' display a sequence of integers with the specified header
50    Sub Display(ByVal results As IEnumerable(Of Integer), _
51       ByVal header As String)
52
```

LINQWithSimpleTypeArray.vb

(3 of 4)

Returns sorted `Integer`s in the array greater than `4`.

`IEnumerable(Of Integer)` object (such as a LINQ query) as a parameter.

Fig. 9.2 | LINQ to Objects using an `Integer` array. (Part 3 of 4.)

Outline

```
53       Console.Write("{0}", header) ' display header
54
55       ' display each element, separated by spaces
56       For Each element In results
57          Console.Write(" {0}", element)
58       Next
59
60       Console.WriteLine() ' add end of line
61    End Sub ' Display
62 End Module ' LINQWithSimpleTypeArray
```

Iterates through the query results.

```
Original array: 2 9 5 0 3 7 1 4 8 5
Array values greater than 4: 9 5 7 8 5
Original array, sorted: 0 1 2 3 4 5 5 7 8 9
Values greater than 4, descending order (separately): 9 8 7 5 5
Values greater than 4, descending order (one query): 9 8 7 5 5
```

LINQWithSimpleTypeArray.vb

(4 of 4)

Fig. 9.2 | LINQ to Objects using an `Integer` array. (Part 4 of 4.)

- `Order By` sorts results in ascending order.
- The `Descending` **query operator** sorts results in descending order.

9.2 Querying an Array Using LINQ

- The **`From` clause** specifies a **range variable** and the data source to query.
- If the condition in the **`Where` clause** evaluates to `True`, the element is subject to the **`Select`** clause.
- The **`Select` clause** specifies what value appears in the results. If omitted, the range variable is selected.

9.2 Querying an Array Using LINQ (Cont.)

- Interfaces define and standardize classes.
- `IEnumerable` is an **interface** describing a collection of objects (such as an array or a LINQ result).
- A `For Each...Next` statement can iterate through any `IEnumerable` object.

Outline

Using LINQ to Query an Array of Employee Objects

- Figure 9.3 presents the `Employee` class.

`Employee.vb`

(1 of 3)

```
1  ' Fig. 9.3: Employee.vb
2  ' Employee class with FirstName, LastName and MonthlySalary properties.
3  Public Class Employee
4     Private firstNameValue As String ' first name of employee
5     Private lastNameValue As String ' last name of employee
6     Private monthlySalaryValue As Decimal ' monthly salary of employee
7
8     ' constructor initializes first name, last name and monthly salary
9     Public Sub New(ByVal first As String, ByVal last As String, _
10       ByVal salary As Decimal)
11
12       FirstName = first
13       LastName = last
14       MonthlySalary = salary
15    End Sub ' New
16
```

Fig. 9.3 | `Employee` class with `FirstName`, `LastName` and `MonthlySalary` properties. (Part 1 of 3.)

Outline

```
17    ' property that gets and sets the employee's first name
18    Public Property FirstName() As String
19       Get
20          Return firstNameValue
21       End Get
22
23       Set(ByVal value As String)
24          firstNameValue = value
25       End Set
26    End Property ' FirstName
27
28    ' property that gets and sets the employee's last name
29    Public Property LastName() As String
30       Get
31          Return lastNameValue
32       End Get
33
34       Set(ByVal value As String)
35          lastNameValue = value
36       End Set
37    End Property ' LastName
```

Employee.vb

(2 of 3)

Fig. 9.3 | `Employee` class with `FirstName`, `LastName` and `MonthlySalary` properties. (Part 2 of 3.)

Outline

```
38
39    ' property that gets and sets the employee's monthly salary
40    Public Property MonthlySalary() As Decimal
41       Get
42          Return monthlySalaryValue
43       End Get
44
45       Set(ByVal value As Decimal)
46          If value >= 0 Then ' if salary is non-negative
47             monthlySalaryValue = value
48          End If
49       End Set
50    End Property ' MonthlySalary
51
52    ' return a String containing the employee's information
53    Public Overrides Function ToString() As String
54       Return String.Format("{0,-10} {1,-10} {2,10:C}", _
55          FirstName, LastName, MonthlySalary)
56    End Function ' ToString
57 End Class ' Employee
```

Employee.vb

(3 of 3)

Fig. 9.3 | `Employee` class with `FirstName`, `LastName` and `MonthlySalary` properties. (Part 3 of 3.)

Outline

LINQWithArrayOfObjects.vb

(1 of 5)

```
1  ' Fig. 9.4: LINQWithArrayOfObjects.vb
2  ' LINQ to Objects using an array of Employee objects.
3  Module LINQWithArrayOfObjects
4     Sub Main()
5        ' initialize array of employees
6        Dim employees As Employee() = { _
7           New Employee("Jason", "Red", 5000D), _
8           New Employee("Ashley", "Green", 7600D), _
9           New Employee("Matthew", "Indigo", 3587.5D), _
10          New Employee("James", "Indigo", 4700.77D), _
11          New Employee("Luke", "Indigo", 6200D), _
12          New Employee("Jason", "Blue", 3200D), _
13          New Employee("Wendy", "Brown", 4236.4D)} ' end initializer list
14
15       Display(employees, "Original array") ' display all employees
16
17       ' filter a range of salaries using AndAlso in a LINQ query
18       Dim between4K6K = _
19          From e In employees _
20          Where e.MonthlySalary >= 4000D AndAlso e.MonthlySalary <= 6000D _
21          Select e
```

The compiler infers that the range variable is of type `Employee`.

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 1 of 5.)

Outline

```
22
23      ' display employees making between 4000 and 6000 per month
24      Display(between4K6K, String.Format( _
25         "Employees earning in the range {0:C}-{1:C} per month", _
26         4000, 6000))
27
28      ' order the employees by last name, then first name with LINQ
29      Dim nameSorted = _
30         From e In employees _
31         Order By e.LastName, e.FirstName _
32         Select e
33
34      Console.WriteLine("First employee when sorted by name:") ' header
35
36      ' attempt to display the first result of the above LINQ query
37      If nameSorted.Count() = 0 Then
38         Console.WriteLine("not found" & vbNewLine)
39      Else
40         Console.WriteLine(nameSorted.First().ToString() & vbNewLine)
41      End If
```

LINQWithArrayOfObjects.vb

(2 of 5)

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 2 of 5.)

Outline

```
42
43       ' use LINQ's Distinct clause to select unique last names
44       Dim lastNames = _
45          From e In employees _
46          Select e.LastName _
47          Distinct
48
49       ' display unique last names
50       Display(lastNames, "Unique employee last names")
51
52       ' use LINQ to select first and last names
53       Dim names = _
54          From e In employees _
55          Select e.FirstName, Last = e.LastName
56
57       Display(names, "Names only") ' display full names
58    End Sub ' Main
59
```

LINQWithArrayOfObjects.vb

(3 of 5)

The compiler creates an anonymous class with the selected properties.

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 3 of 5.)

```
60    ' display a sequence of any type, each on a separate line
61    Sub Display(Of T)(ByVal results As IEnumerable(Of T), _
62       ByVal header As String)
63
64       Console.WriteLine("{0}:", header) ' display header
65
66       ' display each element, separated by spaces
67       For Each element As T In results
68          Console.WriteLine(element)
69       Next
70
71       Console.WriteLine() ' add end of line
72    End Sub ' Display
73 End Module ' LINQWithArrayOfObjects
```

```
Original array:
Jason       Red         $5,000.00
Ashley      Green       $7,600.00
Matthew     Indigo      $3,587.50
James       Indigo      $4,700.77
Luke        Indigo      $6,200.00
Jason       Blue        $3,200.00
Wendy       Brown       $4,236.40

Employees earning in the range $4,000.00-$6,000.00 per month:
Jason       Red         $5,000.00
James       Indigo      $4,700.77
Wendy       Brown       $4,236.40
```

(continued on next page...)

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 4 of 5.)

Outline

`LINQWithArrayOfObjects.vb`

(4 of 5)

Outline

LINQWithArrayOfObjects.vb

(5 of 5)

(continued from previous page…)

```
First employee when sorted by name:
Jason      Blue        $3,200.00

Unique employee last names:
Red
Green
Indigo
Blue
Brown

Names only:
{ FirstName = Jason, Last = Red }
{ FirstName = Ashley, Last = Green }
{ FirstName = Matthew, Last = Indigo }
{ FirstName = James, Last = Indigo }
{ FirstName = Luke, Last = Indigo }
{ FirstName = Jason, Last = Blue }
{ FirstName = Wendy, Last = Brown }
```

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 5 of 5.)

9.2 Querying an Array Using LINQ (Cont.)

- `Count` returns the number of elements in the result.
- The `First` method returns the first element.
- The `Distinct` clause prevents duplicates in results.

9.2 Querying an Array Using LINQ (Cont.)

- In a LINQ `Select` clause, list an object's properties in a comma-separated list.
  - The compiler creates a new class with select properties called an **anonymous class**.
  - Local type inference allows you to use anonymous types without using names.

9.2 Querying an Array Using LINQ (Cont.)

- Overloaded methods can be more compactly coded using a **generic method**.
- Specify a **type parameter list**—placed in parentheses following the method name, begins with keyword `Of` and contains one or more type parameters.
- A **type parameter** is a placeholder for an actual type.
  - When you call a generic method, the compiler infers the type.

Common Programming Error 9.1

If you forget to include the type-parameter list when declaring a generic method, the compiler will not recognize the type parameter names when they're encountered in the method, causing compilation errors.

9.3 Introduction to Collections

- The collection `List(Of T)` can hold a list of whatever type of elements that you want:

```
Dim list1 As List(Of Integer)
Dim list2 As List(Of String)
```

- Classes with this kind of placeholder that can be used with any type are called **generic classes**.

9.3 Introduction to Collections (Cont.)

Method or property	Description
`Add`	Adds an element to the end of the `List`.
`Capacity`	Property that gets or sets the number of elements a `List` can store.
`Clear`	Removes all the elements from the `List`.
`Contains`	Returns `True` if the `List` contains the specified element; otherwise, returns `False`.

Fig. 9.5 | Some methods and properties of class `List(Of T)`. (Part 1 of 2.)

9.3 Introduction to Collections (Cont.)

Method or property	Description
`Count`	Property that returns the number of elements stored in the `List`.
`IndexOf`	Returns the index of the first occurrence of the specified value in the List.
`Insert`	Inserts an element at the specified index.
`Remove`	Removes the first occurrence of the specified value.
`RemoveAt`	Removes the element at the specified index.
`RemoveRange`	Removes a specified number of elements starting at a specified index.
`Sort`	Sorts the `List`.
`TrimToSize`	Sets the `Capacity` of the `List` to the number of elements the `List` currently contains (`Count`).

Fig. 9.5 | Some methods and properties of class `List(Of T)`. (Part 2 of 2.)

Outline

- Figure 9.6 demonstrates dynamically resizing a `List` object.

`ListCollection.vb`

(1 of 3)

```
1  ' Fig. 9.6: ListCollection.vb
2  ' Generic List collection demonstration.
3  Module ListCollection
4     Sub Main()
5        Dim items As New List(Of String) ' create a new List of Strings
6
7        items.Add("red") ' append an item to the List
8        items.Insert(0, "yellow") ' insert the value at index 0
9
10       Console.Write( _
11          "Display list contents with counter-controlled loop:") ' header
12
13       ' display the colors in the list
14       For i = 0 To items.Count - 1
15          Console.Write(" {0}", items(i))
16       Next
17
```

The `Add` method appends its argument to the end of the `List`.

The `Insert` inserts a new element at the specified position.

Displaying the items in the `List`.

Fig. 9.6 | Generic `List` collection demonstration. (Part 1 of 3.)

Outline

ListCollection.vb

(2 of 3)

```
18      ' display colors using For Each...Next in the Display method
19      Display(items, vbNewLine & _
20         "Display list contents with For Each...Next statement:")
21
22      items.Add("green") ' add "green" to the List
23      items.Add("yellow") ' add "yellow" to the List
24      Display(items, "List with two new elements:") ' display the List
25
26      items.Remove("yellow") ' remove the first "yellow"
27      Display(items, "Remove first instance of yellow:") ' display List
28
29      items.RemoveAt(1) ' remove item at index 1
30      Display(items, "Remove second list element (green):") ' display List
31
32      ' check if a value is in the List
33      Console.WriteLine(vbNewLine & """red"" is {0}in the list", _
34         If(items.Contains("red"), String.Empty, "not "))
35
36      ' display number of elements in the List
37      Console.WriteLine(vbNewLine & "Count: {0}", items.Count)
```

More items are added, then the `List` is displayed again.

The `Remove` method removes the first element with a given value.

`RemoveAt` removes the element at the specified index.

`Contains` returns `True` if the `String` is already in the `List`.

`Count` returns the length of the `List`.

Fig. 9.6 | Generic `List` collection demonstration. (Part 2 of 3.)

Outline

```
38
39       ' display the capacity of the List
40       Console.WriteLine("Capacity: {0}", items.Capacity)
41    End Sub ' Main
42
43    ' display the List's elements on the console
44    Sub Display(ByVal items As List(Of String), ByVal header As String)
45       Console.Write(header) ' print header
46
47       ' display each element in items
48       For Each item In items
49          Console.Write(" {0}", item)
50       Next
51
52       Console.WriteLine() ' print end of line
53    End Sub ' Display
54 End Module ' ListCollection
```

```
Display list contents with counter-controlled loop: yellow red
Display list contents with For Each...Next statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Count: 2
Capacity: 4
```

ListCollection.vb

(3 of 3)

The `Capacity` property indicates how many items the `List` can hold without being resized.

Fig. 9.6 | Generic `List` collection demonstration. (Part 3 of 3.)

9.3 Introduction to Collections (Cont.)

- The `Remove` method removes the first element with a given value.
- `RemoveAt` removes the element at the specified index.

9.3 Introduction to Collections (Cont.)

- `Contains` returns `True` if the `String` is already in the `List`.
- `Count` returns the length of the `List`.
- The `Capacity` property indicates how many items the `List` can hold without being resized.

Outline

- Fig. 9.7, a `List` of `String`s is queried.
- LINQ's **deferred execution** means that the query is executed only when the results are retrieved, so the same query can be re-used.

LinqWithListCollection.vb

(1 of 2)

```
1  ' Fig. 9.7: LinqWithListCollection.vb
2  ' LINQ to Objects using a List(Of String).
3  Module LINQWithListCollection
4     Sub Main()
5        ' populate a List of Strings
6        Dim items As New List(Of String)
7        items.Add("aqua") ' add "aqua" to the end of the List
8        items.Add("rust") ' add "rust" to the end of the List
9        items.Add("yellow") ' add "yellow" to the end of the List
10       items.Add("red") ' add "red" to the end of the List
11
12       ' select Strings starting with "r" and convert them to uppercase
13       Dim startsWithR = _
14          From item In items _
15          Where item.StartsWith("r") _
16          Order By item _
17          Select item.ToUpper()
18
```

Selecting `String`s that start with "`r`".

An all-uppercase `String` is returned by `ToUpper`.

Fig. 9.7 | LINQ to Objects using a `List(Of String)`. (Part 1 of 2.)

Outline

```
19       ' display query results
20       For Each item In startsWithR
21          Console.Write("{0} ", item)
22       Next
23
24       Console.WriteLine() ' output end of line
25
26       items.Add("ruby") ' add "ruby" to the end of the List
27       items.Add("saffron") ' add "saffron" to the end of the List
28
29       ' print updated query results
30       For Each item In startsWithR
31          Console.Write("{0} ", item)
32       Next
33
34       Console.WriteLine() ' output end of line
35    End Sub ' Main
36 End Module ' LINQWithListCollection
```

```
RED RUST
RED RUBY RUST
```

LinqWithList Collection.vb

(2 of 2)

Fig. 9.7 | LINQ to Objects using a `List(Of String)`. (Part 2 of 2.)