When LINQ was first introduced to the .NET platform in version 3.5, the C# and VB languages were

each expanded with a large number of new programming constructs used to support the LINQ

technology set. Specifically, the C# language uses the following core LINQ-centric features:

Implicitly typed local variables

Object/collection initialization syntax

Lambda expressions

Extension methods

Anonymous types

static void DeclareImplicitVars()

{

// Implicitly typed local variables.

var myInt = 0;

var myBool = true;

var myString = "Time, marches on...";

// Print out the underlying type.

Console.WriteLine("myInt is a: {0}", myInt.GetType().Name);

Console.WriteLine("myBool is a: {0}", myBool.GetType().Name);

Console.WriteLine("myString is a: {0}", myString.GetType().Name);

}

This language feature is very helpful, and often mandatory, when using LINQ. As you will see during

this chapter, many LINQ queries will return a sequence of data types, which are not known until compile

time. Given that the underlying data type is not known until the application is compiled, you obviously

can’t declare a variable explicitly!

static void LambdaExpressionSyntax()

{

// Make a list of integers.

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

list.AddRange(new int[] { 20, 1, 4, 8, 9, 44 });

// C# lambda expression.

List<int> evenNumbers = list.FindAll(i => (i % 2) == 0);

Console.WriteLine("Here are your even numbers:");

foreach (int evenNumber in evenNumbers)

{

Console.Write("{0}\t", evenNumber);

}