Using LINQ with

341.6k 8 17

In this tutorial, you'll learn basics of LINQ, types in LINQ, and how to use LINQ in C#.

Writing software means that you need to have a database sitting at the back end, and most of the time goes into writing queries to retrieve and manipulate data. Whenever someone talks about data, we tend to only think of the information that is contained in a relational database or in an XML document.

The kind of data access that we had prior to the release of .NET 3.5 was only meant for or limited to accessing data that resides in traditional data sources as the two just mentioned. But with the release of .NET 3.5 and higher versions like .NET 4.0 and .NET 4.5, that has Language INtegrated Query (LINQ) incorporated into it, it is now possible to deal with data residing beyond the traditional homes of information storage. For instance, you can query a generic List type containing a few hundred integer values and write a LINQ expression to retrieve the subset that meets your criterion, for example, either even or odd.

The LINQ feature, as you may have gathered, was one of the major differences between .NET 3.0 and .NET 3.5. LINQ is a set of features in Visual Studio that extends powerful query capabilities into the language syntax of C# and VB .NET

LINQ introduces a standard, unified, easy-to-learn approach for querying and modifying data, and can be extended to support potentially any type of data store. Visual Studio also supports LINQ provider assemblies that enable the use of LINQ queries with various types of data sources including relational data, XML, and in-memory data structures.

In this article, I will cover the following:

1. Introduction to LINQ
2. Architecture of LINQ
3. Using LINQ to Objects
4. Using LINQ to SQL
5. Using LINQ to XML

LINQ is an innovation that Microsoft made with the release of V 08 and .NET Framework version 3.5 that promises to revolutionize the way that developers have been

the recent releases of .NET 4.0/4.5 and Visual Studio 2012. As I mentioned previously, LINQ introduces the standard and unified concept of querying various types of data sources falling in the range of relational databases, XML documents, and even in-memory data structures. LINQ supports all these types of data stores using LINQ query expressions of first-class language constructs in C#. LINQ offers the following advantages:

- LINQ offers common syntax for querying any type of data source; for example, you can query an XML document in the same way as you query a SQL database, an ADO.NET dataset, an in-memory collection, or any other remote or local data source that you have chosen to connect to and access by using LINQ.
- LINQ bridges the gap and strengthens the connection between relational data and the object-oriented world.
- LINQ speeds development time by catching many errors at compile time and including IntelliSense and debugging support.
- LINQ query expressions (unlike traditional SQL statements) are strongly typed.

The LINQ assemblies provide all the functionality of accessing various types of data stores under one umbrella. The core LINQ assemblies are listed in Table 1-1.

Table 1-1. Core LINQ Assemblies

Assembly Name	Description
System.LINQ	Provides classes and interfaces that support LINQ queries
System.Collections.Generic	Allows users to create strongly typed collections that provide better type safety and performance than nongeneric strongly typed collections (LINQ to Objects)
System.Data.LINQ	Provides the functionality to use LINQ to access relational databases (LINQ to SQL)
System.XML.LINQ	Provides functionality for accessing XML documents using LINQ (LINQ to XML)
System.Data.Linq.Mapping	Designates a class as an entity class associated with a database

Architecture of LINQ

LINQ consists of the following three major components:

- LINQ to Objects
- LINQ to ADO.NET, that includes
- LINQ to SQL (formerly called DLinq)
- LINQ to DataSets (formerly called LINQ over DataSets)
- LINQ to Entities
- LINQ to XML (formerly called XLinq)

Figure 1-1 depicts the LINQ architecture, that clearly shows the various components of LINQ and their related data stores.

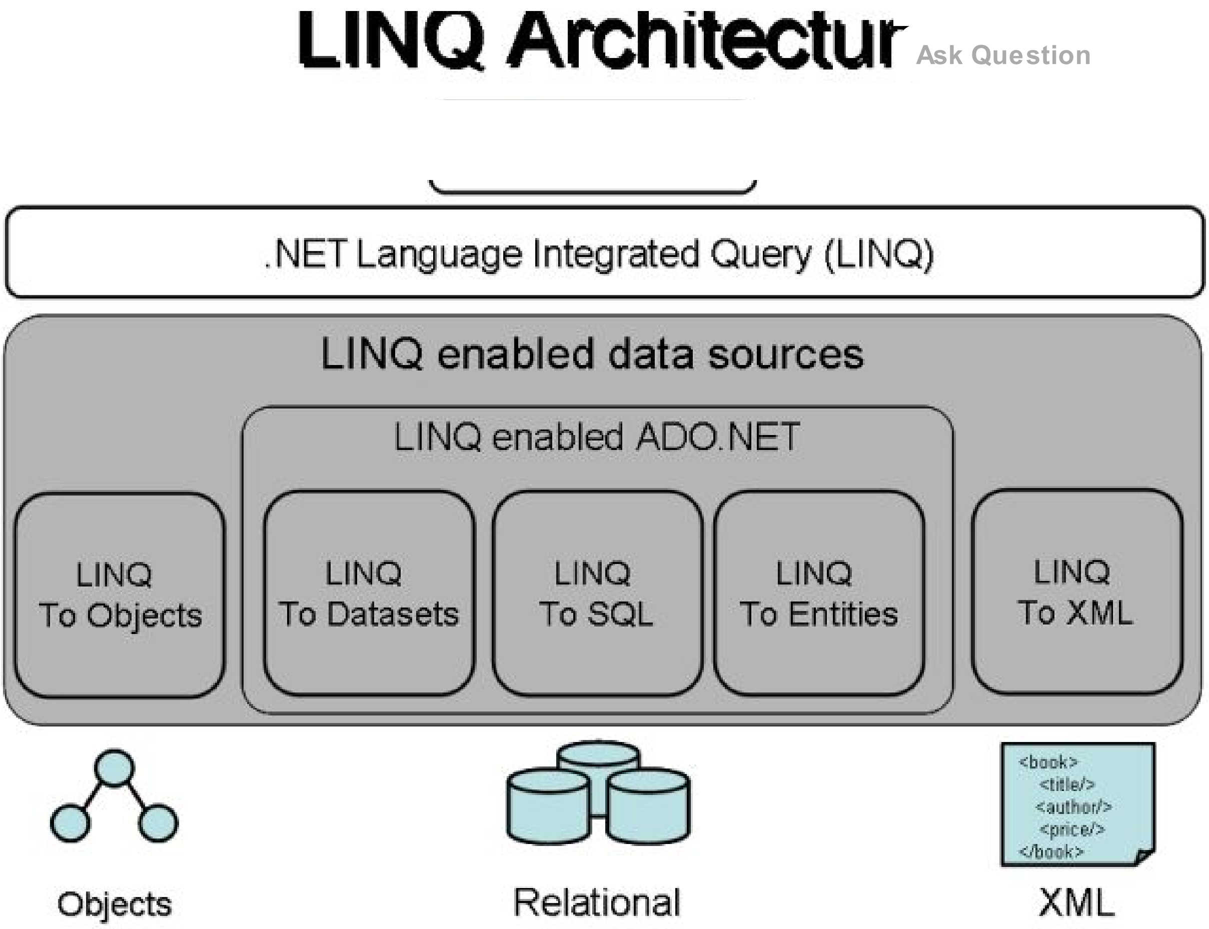


Figure 1-1. LINQ architecture

LINQ to Objects deals with in-memory data. Any class that implements the IEnumerable<T> interface (in the System.Collections.Generic namespace) can be queried with Standard Query Operators (SQOs).

LINQ to ADO.NET (also known as LINQ-enabled ADO .NET) deals with data from external sources, basically anything ADO.NET can connect to. Any class that implements IEnumerable<T> or IOueryable<T> (in the System.Linq namespace) can be queried with SQOs. The LINQ to ADO.NET functionality can be done by using the System. Data.Linq namespace.

LINQ to XML is a comprehensive API for in-memory XML programming. Like the rest of LINQ, it includes SQOs, and it can also be used in concert with LINQ to ADO.NET, but its primary purpose is to unify and simplify the kinds of things that disparate XML tools, such as XQuery XPath, and XSLT, are typically used to do. The LINQ to XML functionality can be done by using the System.Xml.Linq namespace.

In this article, we'll work with the three techniques LINQ to Objects, LINQ to SQL, and LINQ to DataSets.

Using LINQ to Objects

The term LINQ to Objects refers to the use of LINQ queries to access in-memory data structures. You can query any type that supports IEnumerable<T>. This means that you can use LINQ

System.Reflection classes to return information about types st ied assembly, and then filter those results using LINQ. Or you can import text files into enumerable data structures

over traditional foreach loops:

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

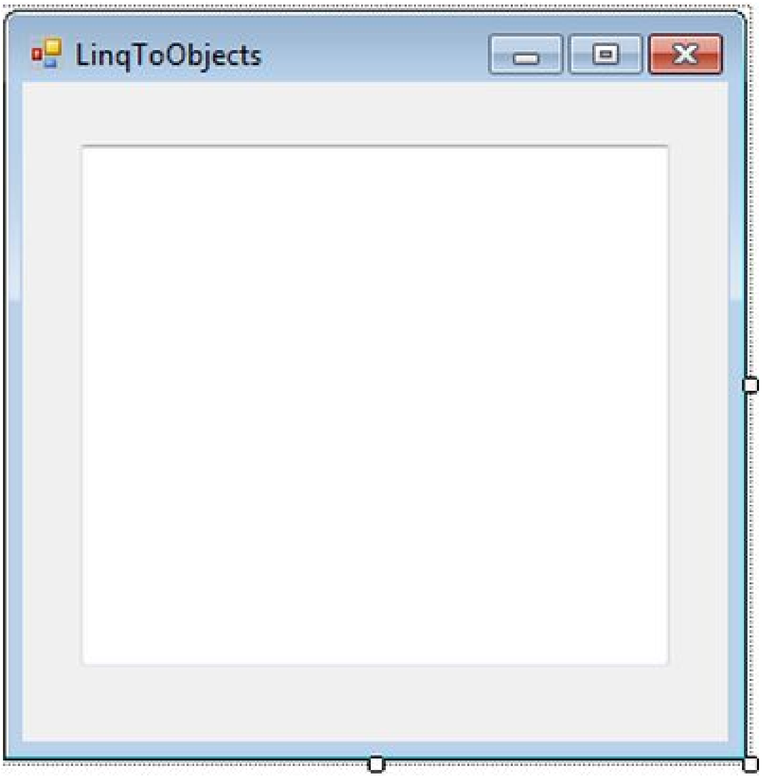
In general, the more complex the operation you want to perform on the data, the greater the benefit you will realize using LINQ as opposed to traditional iteration techniques.

Try It Out: Coding a Simple LINQ to Objects Query

In this exercise, you'll create a Windows Forms Application having one Text Box. The application will retrieve and display some names from an array of strings in a TextBox control using LINQ to Objects.

1. Open Visual Studio 2012 and select "File" -> "New" -> "Project...".
2. Choose "Windows Forms" project.
3. Right-click the "Form1.cs" in the solution, select "Rename" and rename the form to "LinqToObjects".
4. Drag a Text Box control onto the form, and position it towards the center of the Form. Select this Text Box and navigate to the Properties window.

   Now your LinqToObjects form in Design view should be such as shown in Figure 1-2.

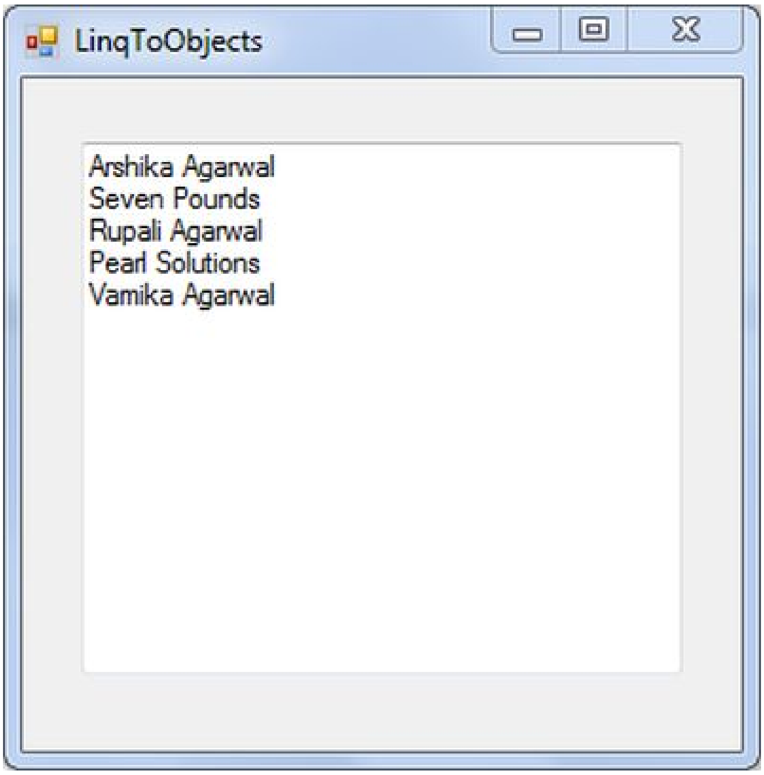


5. Now double-click on the empty surface of the "LinqToObje d it will open the code editior window, showing the "LinqToObject_Load" event. Place the following code in

Listing 1-1. LinqToObjects.cs

```
//Define string array
string[] names = { "Life is Beautiful",
                              "Arshika Agarwal",
                              "Seven Pounds",
                              "Rupali Agarwal",
                              "Pearl Solutions",
                              "Vamika Agarwal",
                              "Vidya Vrat Agarwal",
                              "C-Sharp Corner Mumbai Chapter"
                           };
//Linq query
IEnumerable<string> namesOfPeople = from name in names
                                    where name.Length <= 16
                                    select name;
foreach (var name in namesOfPeople)
{
    txtDisplay.AppendText(name+"\n");
}
```

Run the program by pressing "Ctrl+F5", and you should see the results shown in Figure 1-3.



```
Arshika Agarwal
Seven Pounds
Rupali Agarwal
Pearl Solutions
Vamika Agarwal
```

Figure 1-3. Retrieving names from a string array using LINQ to Objects

Using LINQ to SQL

many operations. It connects to a database, converts LINQ con L, submits the SQL, transforms results into objects, and even tracks changes and automatically requests database

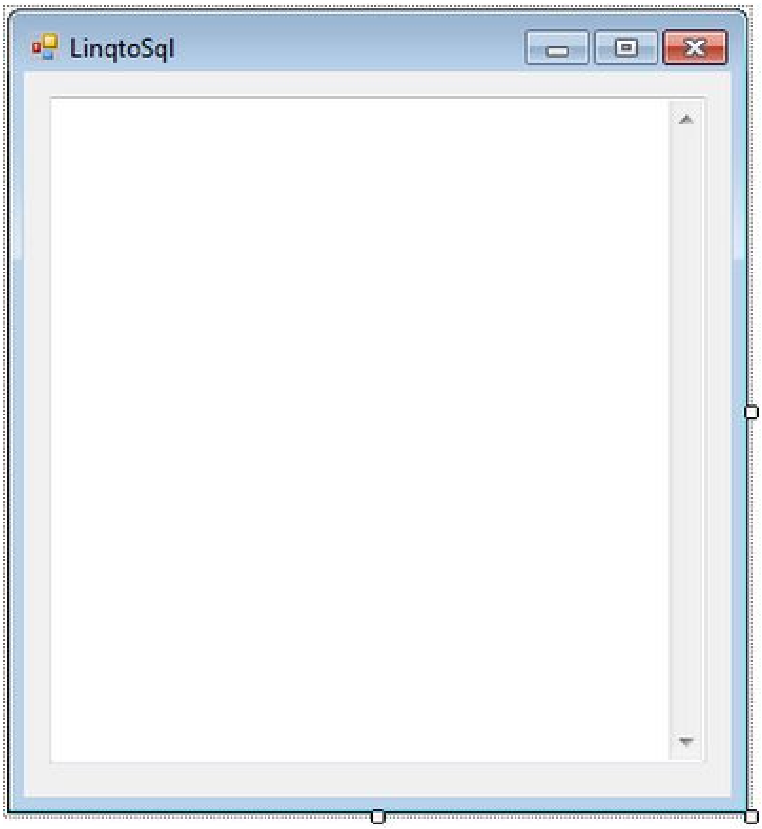
A simple LINQ query requires three things:

- Entity classes
- A data context
- A LINQ query

Try It Out: Coding a Simple LINQ to SQL Query

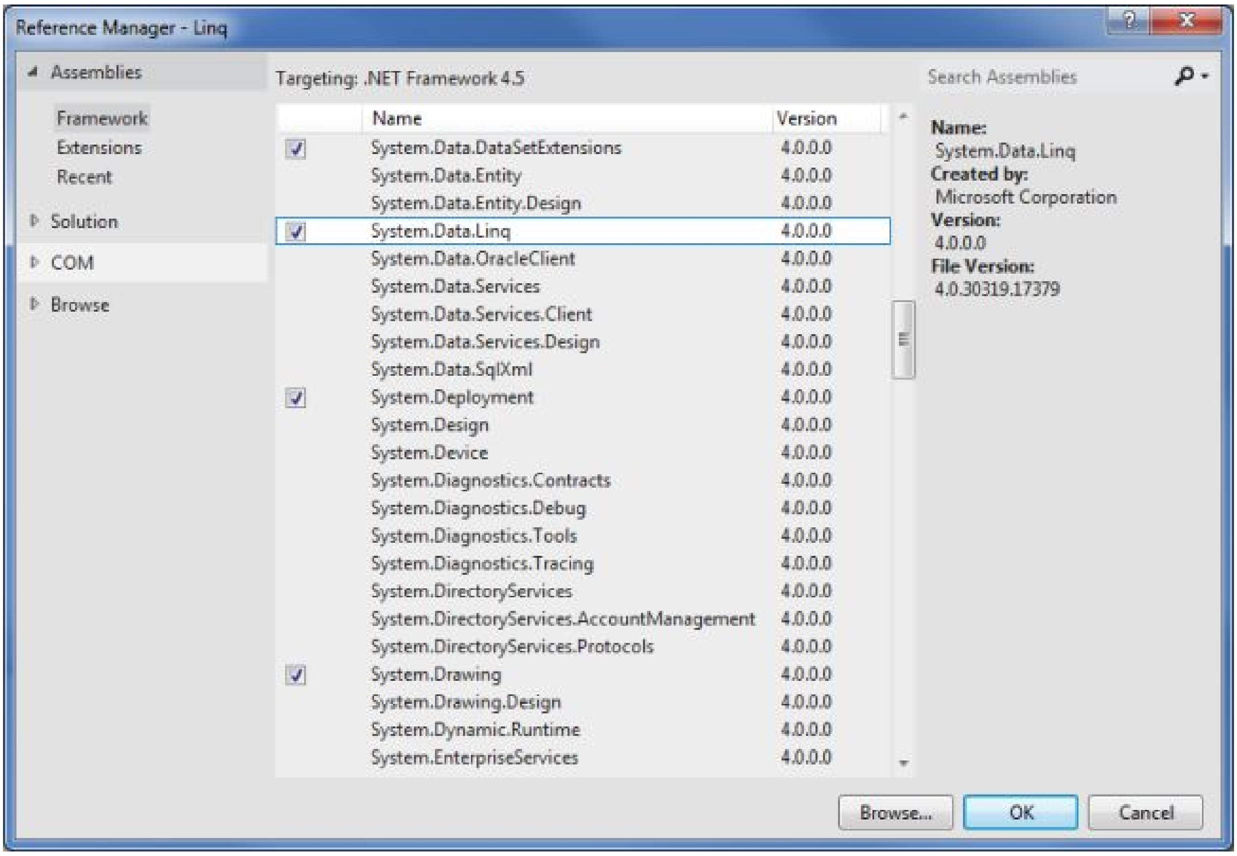
In this exercise, you'll use LINQ to SQL to retrieve all contact details from the AdventureWorks Person.Contact table.

1. Navigate to Solution Explorer, right-click your LINQ project and select "Add Windows Form". In the "Add New Item" dialog make sure "Windows Form" is selected and then rename the "Form1.cs" to "LinqToSql". Click "Add".
2. Drag a Text Box control onto the form, and position it at towards the center of the Form. Select this Text Box and navigate to the Properties window, and set the following properties:
    - Name to txtLinqToSql.
    - Multiline to True.
    - ScrollBars to Vertical.
3. Now your LinqToSql form in Design view should be such as shown in Figure 1-4.



4. Before we begin with coding the functionality, we must ad[illegible] assembly references. LinqToSql will require an assemble reference of System.Data.Linq to be added

To do so, in Solution Explorer, select the "References", right-click and choose "Add Reference". From the opened Reference Manager dialog, scroll down to the assembly list select System.Data.Linq and check the checkbox shown in front of it as shown in figure 1-5 and click "OK".



Reference Manager - Linq

Assemblies: Framework, Extensions, Recent; Solution; COM; Browse

Targeting: .NET Framework 4.5

	Name	Version
☑	System.Data.DataSetExtensions	4.0.0.0
	System.Data.Entity	4.0.0.0
	System.Data.Entity.Design	4.0.0.0
☑	System.Data.Linq	4.0.0.0
	System.Data.OracleClient	4.0.0.0
	System.Data.Services	4.0.0.0
	System.Data.Services.Client	4.0.0.0
	System.Data.Services.Design	4.0.0.0
	System.Data.SqlXml	4.0.0.0
☑	System.Deployment	4.0.0.0
	System.Design	4.0.0.0
	System.Device	4.0.0.0
	System.Diagnostics.Contracts	4.0.0.0
	System.Diagnostics.Debug	4.0.0.0
	System.Diagnostics.Tools	4.0.0.0
	System.Diagnostics.Tracing	4.0.0.0
	System.DirectoryServices	4.0.0.0
	System.DirectoryServices.AccountManagement	4.0.0.0
	System.DirectoryServices.Protocols	4.0.0.0
☑	System.Drawing	4.0.0.0
	System.Drawing.Design	4.0.0.0
	System.Dynamic.Runtime	4.0.0.0
	System.EnterpriseServices	4.0.0.0

Search Assemblies

Name:
System.Data.Linq
Created by:
Microsoft Corporation
Version:
4.0.0.0
File Version:
4.0.30319.17379

Figure 1-5. Adding LINQ References

5. Open the newly added form "LinqToSql.cs" in code view. Add the code shown in Listing 1-2 to LinqToSql.cs.

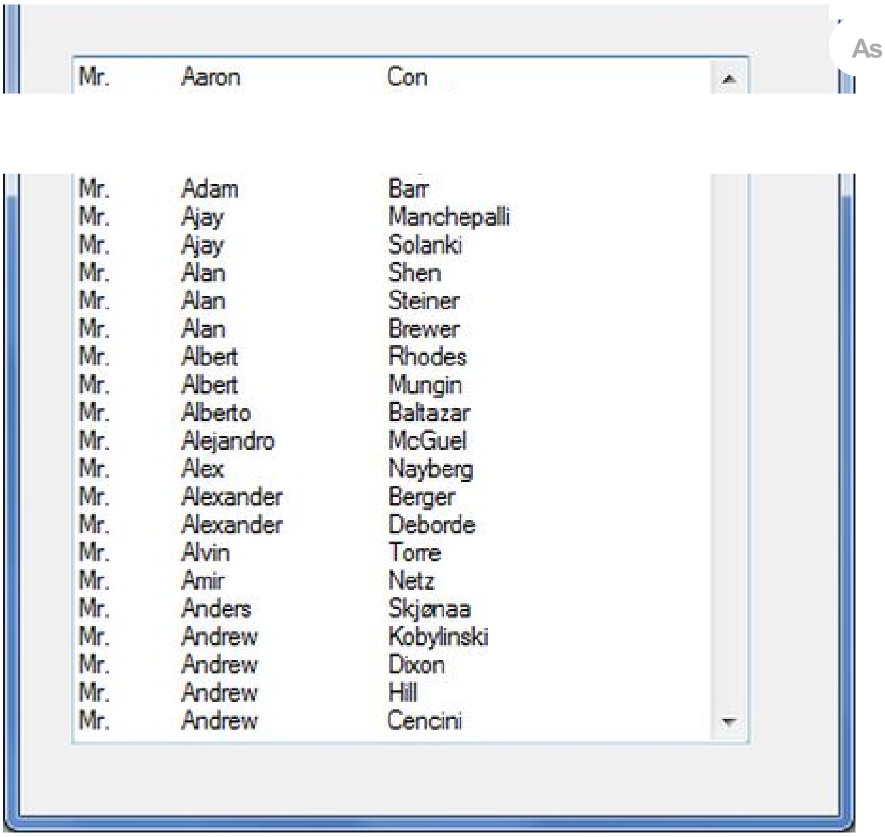
Listing 1-2. LinqToSql.cs

```
// Must add these two namespaces for LinqToSql
using System.Data.Linq;
using System.Data.Linq.Mapping;
[Table(Name = "Person.Person")]
         public class Contact
         {
              [Column]
              public string Title;
              [Column]
              public string FirstName;
              [Column]
              public string LastName;
{
    // connection string
    string connString = @"server = .\sql2012;integrated securi

        // Create data context
        DataContext db = new DataContext(connString);
        // Create typed table
        Table<Contact> contacts = db.GetTable<Contact>();
        // Query database
        var contactDetails =
           from c in contacts
           where c.Title == "Mr."
           orderby c.FirstName
           select c;
        // Display contact details
        foreach (var c in contactDetails)
        {
            txtLinqtoSql.AppendText(c.Title);
            txtLinqtoSql.AppendText("\t");
            txtLinqtoSql.AppendText(c.FirstName);
            txtLinqtoSql.AppendText("\t");
            txtLinqtoSql.AppendText(c.LastName);
            txtLinqtoSql.AppendText("\n");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

6. Now, to set the LinqToSql form as the startup form, open Program.cs in the code editor and modify the following:

   Application.Run(new LinqToObjects());
   to appear as:
   Application.Run(new LinqToSql());.
7. Build the solution and then run the program by pressing Ctrl+F5, and you should see the results shown in Figure 1-6.



```
Mr.    Aaron        Con
Mr.    Adam         Barr
Mr.    Ajay         Manchepalli
Mr.    Ajay         Solanki
Mr.    Alan         Shen
Mr.    Alan         Steiner
Mr.    Alan         Brewer
Mr.    Albert       Rhodes
Mr.    Albert       Mungin
Mr.    Alberto      Baltazar
Mr.    Alejandro    McGuel
Mr.    Alex         Nayberg
Mr.    Alexander    Berger
Mr.    Alexander    Deborde
Mr.    Alvin        Torre
Mr.    Amir         Netz
Mr.    Anders       Skjønaa
Mr.    Andrew       Kobylinski
Mr.    Andrew       Dixon
Mr.    Andrew       Hill
Mr.    Andrew       Cencini
```

Figure 1-6. Retrieving contact details with LINQ to SQL

How It Works

You define an entity class, Contact as in the following:

```
[Table(Name = "Person.Person")]
        public class Contact
        {
            [Column]
            public string Title;
            [Column]
            public string FirstName;
            [Column]
            public string LastName;
        }
```

Entity classes provide objects in which LINQ stores data from data sources. They're like any other C# class, but LINQ defines attributes that tell it how to use the class.

The [Table] attribute marks the class as an entity class and has an optional Name property that can be used to provide the name of a table, that defaults to the class name. That's why you name the class Contact rather than Person.Contact.

[Table(Name = "Person.Contact")]

public class Contact and then you'd need to change the typed table definition to:

```
Table<Contact> contacts = db.GetTable<Contact>();
```

The [Column] attribute marks a field as one that will hold data f u can declare fields in an entity class that don't map to table columns, and LINQ will just ignore them, but

the default names do not need to be identical in case to the names used in the database.)

You create a data context as in the following:

```
// Create data context
DataContext db = new DataContext(connString);
```

A data context does what an ADO.NET connection does, but it also does things that a data provider handles. It not only manages the connection to a data source, but also translates LINQ requests (expressed in SQO) into SQL, passes the SQL to the database server, and creates objects from the result set.

You create a typed table as in the following:

```
// Create typed table
Table<Contact> contacts = db.GetTable<Contact>();
```

A typed table is a collection (of type System.Data.Linq.Table<T>) whose elements are of a specific type. The GetTable method of the DataContext class tells the data context to access the results and indicates where to put them. Here, you get all the rows (but only three columns) from the Person.Contact table, and the data context creates an object for each row in the contacts typed table.

You declare a C# 2012 implicitly typed local variable, contactDetails, of type var:

```
// Query database
var contactDetails =
```

An implicitly typed local variable is just what its name implies. When C# sees the var type, it infers the type of the local variable based on the type of the expression in the initializer to the right of the = sign.

You initialize the local variable with a query expression as in the following:

```
from c in contacts
where c.Title == "Mr."
orderby c.FirstName
select c;
```

A query expression is composed of a from clause and a query body. You use a WHERE condition in the query body here. The from clause declares an iteration variable, c, to be used to iterate over the result of the expression, contacts, that is, over the typed table you earlier created and loaded. In each iteration it will select the rows that meets the WHERE clause. In other words Title must be "Mr.".

versions like C#2012.

```
{
            txtLinqtoSql.AppendText(c.Title);
            txtLinqtoSql.AppendText("\t");
            txtLinqtoSql.AppendText(c.FirstName);
            txtLinqtoSql.AppendText("\t");
            txtLinqtoSql.AppendText(c.LastName);
            txtLinqtoSql.AppendText("\n");
}
```

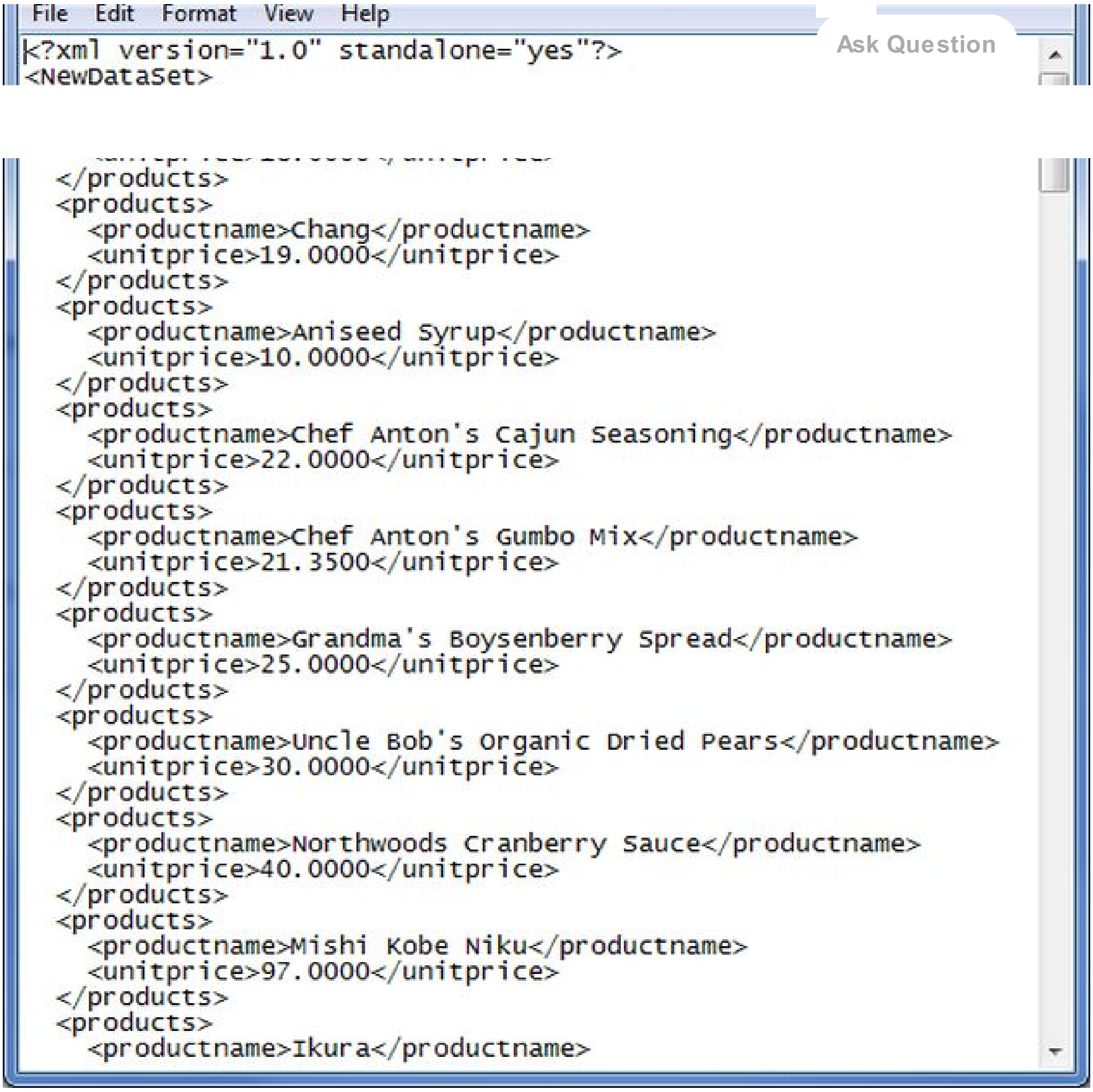
Despite the new C# 2008 features and terminology, this will still feel familiar. Once you get the hang of it, it's an appealing alternative for coding queries. You basically code a query expression instead of SQL to populate a collection that you can iterate through with a foreach statement. However, you provide a connection string, but don't explicitly open or close a connection. Further, no command, data reader, or indexer is required. You don't even need the System.Data or System.Data.SqlClient namespaces to access SQL Server.

Pretty cool, isn't it?

Using LINQ to XML

LINQ to XML provides an in-memory XML programming API that integrates XML querying capabilities into C# 2012 to take advantage of the LINQ framework and add query extensions specific to XML. LINQ to XML provides the query and transformation power of XQuery and XPath integrated into .NET.

From another perspective, you can also think of LINQ to XML as a full-featured XML API comparable to a modernized, redesigned SystemXml API plus a few key features from XPath and XSLT. LINQ to XML provides facilities to edit XML documents and element trees in memory, as well as streaming facilities. A sample XML Document looks as in Figure 1-7.



```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
```

```
      <unitprice>18.0000</unitprice>
  </products>
  <products>
    <productname>Chang</productname>
    <unitprice>19.0000</unitprice>
  </products>
  <products>
    <productname>Aniseed Syrup</productname>
    <unitprice>10.0000</unitprice>
  </products>
  <products>
    <productname>Chef Anton's Cajun Seasoning</productname>
    <unitprice>22.0000</unitprice>
  </products>
  <products>
    <productname>Chef Anton's Gumbo Mix</productname>
    <unitprice>21.3500</unitprice>
  </products>
  <products>
    <productname>Grandma's Boysenberry Spread</productname>
    <unitprice>25.0000</unitprice>
  </products>
  <products>
    <productname>Uncle Bob's Organic Dried Pears</productname>
    <unitprice>30.0000</unitprice>
  </products>
  <products>
    <productname>Northwoods Cranberry Sauce</productname>
    <unitprice>40.0000</unitprice>
  </products>
  <products>
    <productname>Mishi Kobe Niku</productname>
    <unitprice>97.0000</unitprice>
  </products>
  <products>
    <productname>Ikura</productname>
```

Figure 1-7. XML Document

Try It Out: Coding a Simple LINQ to XML Query

In this exercise, you'll use LINQ to XML to retrieve element values from an XML document.

1. Navigate to Solution Explorer, right-click the LINQ project, and and select Windows Form. In the opened "Add New Item" dialog make sure Windows Form is selected and then rename the "Form1.cs" to "LinqToXml". Click "Add".
2. Drag a Text Box control onto the form, and position it towards the center of the form. Select this Text Box and navigate to the Properties window.
3. Now your LinqToXml form in Design view should look such as shown in Figure 1-8.

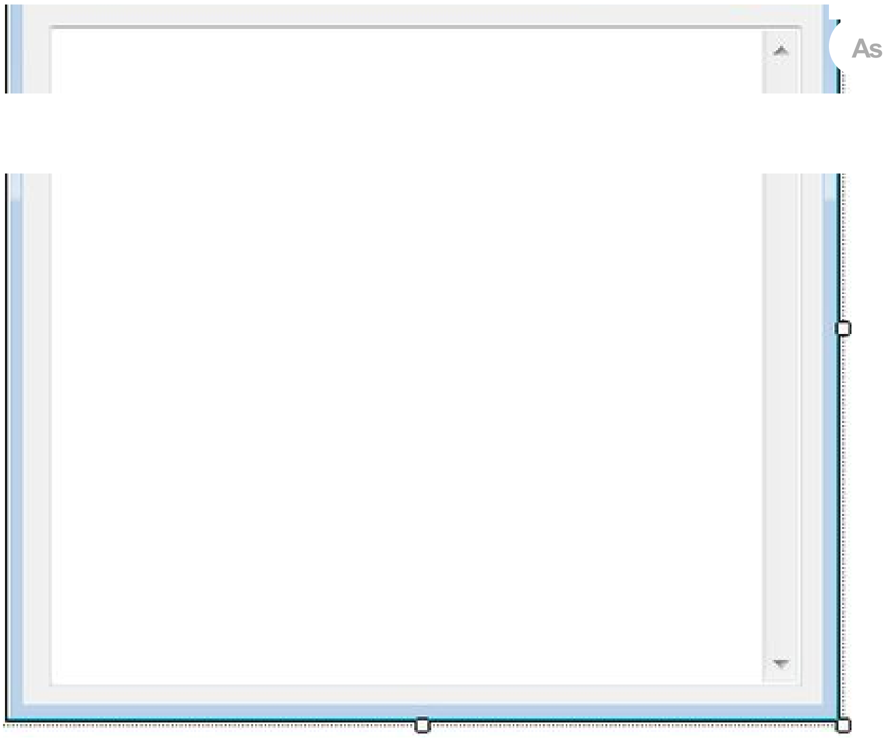


Figure 1-8. Design view of LinqToXml form

4. Open the newly added form "LinqToXml.cs" in code view. Add the code shown in Listing 1-3 in "LinqToXml.cs".

   Listing 1-3. LinqToXml.cs

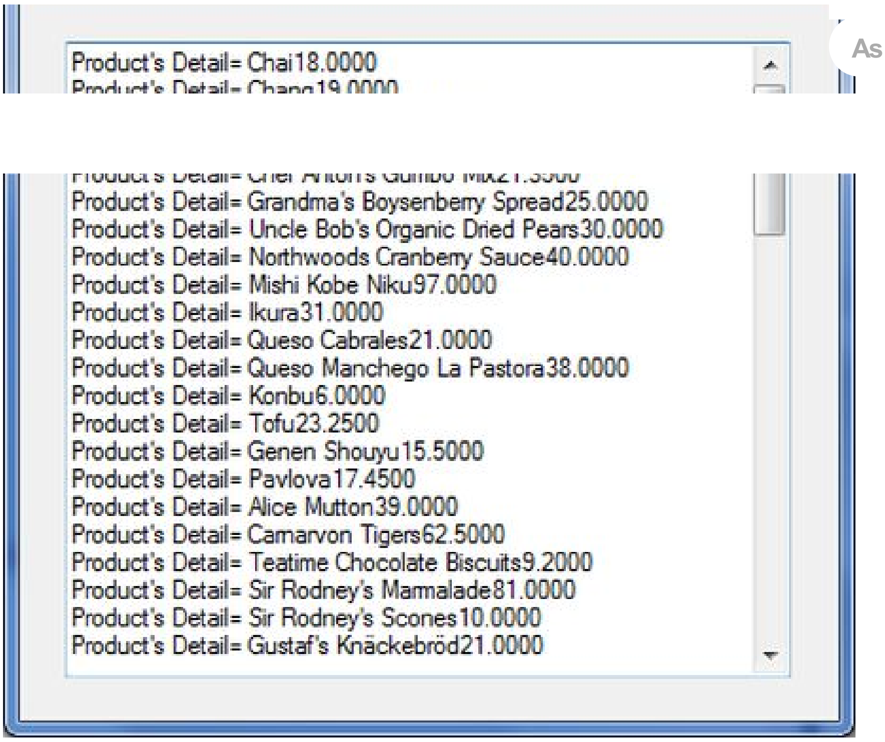
```
using System.Xml.Linq;
//Load the productstable.xml in memory
XElement doc = XElement.Load(@"C:\VidyaVrat\C#\Linq\productstable.xml"
//Query xml doc
var products = from prodname in doc.Descendants("products")

              select prodname.Value;
//Display details
foreach (var prodname in products)
{
    txtLinqToXml.AppendText("Product's Detail= ");
    txtLinqToXml.AppendText(prodname);
    txtLinqToXml.AppendText("\n");
```

5. Now, to set the LinqToSql form as the startup form, open the Program.cs in code editor and modify the:

   Application.Run(new LinqToSql());
   to appear as:
   Application.Run(new LinqToXml());.

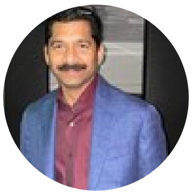
6. Build the solution, and then run the program by pressing "Ctrl+F5" and you should see the results as shown in Figure 1-9.



```
Product's Detail= Chai18.0000
Product's Detail= Chang19.0000
Product's Detail= Chef Anton's Gumbo Mix21.3500
Product's Detail= Grandma's Boysenberry Spread25.0000
Product's Detail= Uncle Bob's Organic Dried Pears30.0000
Product's Detail= Northwoods Cranberry Sauce40.0000
Product's Detail= Mishi Kobe Niku97.0000
Product's Detail= Ikura31.0000
Product's Detail= Queso Cabrales21.0000
Product's Detail= Queso Manchego La Pastora38.0000
Product's Detail= Konbu6.0000
Product's Detail= Tofu23.2500
Product's Detail= Genen Shouyu15.5000
Product's Detail= Pavlova17.4500
Product's Detail= Alice Mutton39.0000
Product's Detail= Carnarvon Tigers62.5000
Product's Detail= Teatime Chocolate Biscuits9.2000
Product's Detail= Sir Rodney's Marmalade81.0000
Product's Detail= Sir Rodney's Scones10.0000
Product's Detail= Gustaf's Knäckebröd21.0000
```

Figure 1-9. Retrieving product details with LINQ to XML

Architecture of LINQ LINQ LINQ in C# Using LINQ to Objects Using LINQ to SQL Using LINQ to XML



Vidya Vrat Agarwal TOP 100

I am a Principal Software Engineering Manager in the Microsoft Azure-CXP team. I am a Software Architect, Author, Technical Reviewer, Blogger, Speaker, and a Mentor. I am an Ex-Microsoft MVP, and a C# Corner MVP. I am a... Read more

http://www.mypassionfor.net

66 **11.6m** **6** **7**

17 **8**

Rajanikant Hawaldar

35 37.6k 249.1k 1 0 Reply

SO GOOD!

sparkle dai Oct 28, 2013

2153 5 0 1 0 Reply

Simple and easily understandable thank you very much for sharing the knowledge on LINQ. Good examples.

Srinubabu Ravilla Oct 28, 2013

1476 693 846.3k 1 0 Reply

could you help me how i can learn something called ORM used to generate classes when i use linq in my asp.net web app ?!

HUSSAM KLHASAN Oct 23, 2013

2151 7 0 2 1 Reply

I am already working on an article on ADO .NET Entity Framework which is Microsoft's ORM.

Vidya Vrat Agarwal Oct 23, 2013

66 28.7k 11.6m 1

I downloaded your book beginning in c# 5.0 database and started to study it ,,,,,

HUSSAM KLHASAN Oct 23, 2013

2151 7 0 1 0 Reply

Good Article.

Gomathi Palaniswamy Oct 23, 2013

486 4.8k 2.1m 1 0 Reply

good work ,thank you

HUSSAM KLHASAN Oct 22, 2013

2151 7 0 1 0 Reply

FEATURED ARTICLES

ChatGPT Completions With C#

F# Records

How to Use Change Tokens In .NET 7

Microsoft Teams Incoming Webhook Integration With ASP.NET Core

Static Abstract Interface Members In C# 11 And Curiously Recurring Template Pattern

React Skill

Ask Question

GET CERTIFIED

HTML5