



CREATIVE DATA TECHNOLOGIES, INC.

[DATALAYER.NET](http://DATALAYER.NET)<sup>™</sup>

---

# Getting Started Guide

# Table of Contents

<b>Table of Contents.....</b>	<b>1</b>
<b>Chapter 1 - Installation .....</b>	<b>2</b>
1.1 Installation Steps.....	2
<b>Chapter 2 – DataLayer.NET Overview .....</b>	<b>3</b>
2.1 Overview of DataLayer.NET Components .....	3
2.2 Summary.....	4
<b>Chapter 3 – Quick Start Sample Application .....</b>	<b>5</b>
3.1 Project Source Code Organization.....	5
3.2 SQL Server Northwind sample database required.....	5
3.3 High level steps to create the Quick Start Application.....	6
3.4 Steps to create the Quick Start Application .....	6
3.5 Summary.....	14



# Chapter 1 - Installation

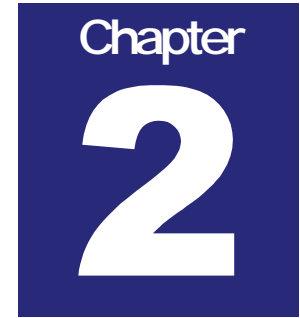
## 1.1 Installation Steps

If you have already downloaded the ZIP file containing the DataLayer.NET Library, you have already received the instructions below to follow for installing the product:

1. Download the Zip file using the hyperlink contained in the email that was sent to you to your local hard drive in a temporary folder (anywhere you like is fine).
2. Create a Folder called "DataLayer.NET" in your C:\Program Files folder (so it will be C:\Program Files\DataLayer.NET).

Note: A Setup program was not created, on purpose, so that you, as developers, would not have to worry about malicious activity from a Setup.exe (or .msi) from an unknown or untrusted party).

3. Unzip the contents of the Zip file to the new folder you just created.
4. Create a desktop shortcut icon to the C:\Program Files\DataLayer.NET\DataLayerCodeGenerator.exe program.
5. That's it. The DataLayer.dll file is the one main assembly you will be adding as a reference in all of your projects.



## Chapter 2 - DataLayer.NET Overview

### 2.1 Overview of DataLayer.NET Components

The DataLayer.NET Library is broken into two main components. The first component is called the **DataConnection** class. It is used to help you manage the database connection, manage transactions (if needed) and it provides scalar database functions for your use. The second component is called the **DataHandler** class. This is the main component that does most of the database work for you in data retrieval and managed updates.

Here is the design philosophy we used when we built the DataLayer.NET Library:

- § Make the components intuitive and easily understood by most programmers.
- § Provide functionality that saves the programmer from having to work at such a low level with the ADO.NET Framework (directly interacting with ADO.NET classes such as the SqlConnection, SqlDataAdapter, SqlDataReader, SqlTransaction, SqlCommand, SqlParameter, and all the equivalent classes for the ODBC and OLE-DB interfaces).
- § Provide a programming platform that is independent of any particular back-end database, where none of the interaction with the ADO.NET classes above will need to be coded by the programmer, making it much easier to scale an application from SQL Server to Oracle (OLE-DB driver), for example.
- § Make the library capable of accessing any of the following three types of databases:
  - \* Microsoft SQL Server
  - \* Any ODBC database
  - \* Any OLE-DB database
- § Handle NULLS for the VB programmer. When reading or setting database values, VB programmers do not have the ability to represent a NULL value with program variables. The DataLayer.NET has a feature you can enable or disable that will automatically convert null values to each data type's MinValue constant. For example, a NULL integer will be read in as Integer.MinValue, and a NULL datetime value will be read in as DateTime.MinValue.
- § Dramatically decrease the amount of coding that is required to create typical database interactive programs.
- § Provide support for Parameterized SQL statements for best practices. The use of parameterized SQL protects your applications from SQL Injection attacks, and the SQL also

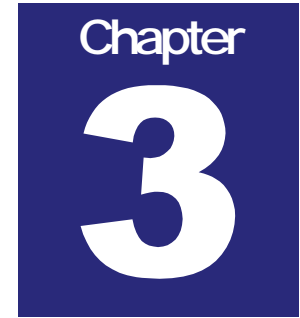
runs much faster, as the back-end database (particularly SQL Server) can cache the SQL command signatures on the cache and run the SQL much faster upon subsequent matching SQL execution.

- § Fully support the use of the DataLayer.NET library for VB.NET programmers as well as C#.NET programmers.
- § Provide a strongly typed interface to the column data, exposing the actual database column types read from the database.
- § Leverage the IntelliSense feature of the Visual Studio.NET environment to save typing time and avoid typing errors during programming. This is an additional benefit of the strongly typed interface of the DataLayer.NET library.
- § Trust and empower the programmers, but do not tie their hands behind their backs. Provide them with the data access objects to make their lives easier, but do not prevent them from directly sending some SQL to the database should they see fit. Extend their SQL Knowledge; do not try to replace it.
- § From the programmer's perspective, the library must be flexible to changes in the database structures (tables). The programmer will only have to update the schema information in a single class (the inherited DataHandler class for the table).
- § Performance must be a key goal in the design, as programmers will not want to use the library if they find out that there is a 10 – 20% reduction in performance by doing so. On the contrary, our clients experience a significant increase in performance because they adopt the parameterized SQL methodology as well as using the DataLayer.NET library.
- § Provide a record retrieval system that can either retrieve ALL of the rows of the result set into a buffer (DataSet), or retrieve the rows one page at a time, where page size and which page to retrieve can be specified by the user at runtime.

## 2.2 Summary

You can see from the above list of features above, the DataLayer.NET is going to save you a lot of coding and make your applications a lot easier to create and maintain. Some customers are stating that they are saving as much as 60 to 70% code savings in the data access layers of their programs.

In the next chapter for this Quick Start Guide, you will be led through the construction of a very simplistic sample application to give you an idea how to start using the DataLayer.NET library.



# Chapter 3 – Quick Start Sample Application

## 3.1 Project Source Code Organization

As you create new programs using Visual Studio.NET, the default location where projects are stored is in a folder name under your “My Documents” folder called “Visual Studio Projects” (or “Visual Studio 2005\Projects” if you are using the new Visual Studio 2005). The physical path to these folders can get quite long, so we suggest simply creating a directory called “Projects” directly at the root of your C: drive (i.e. C:\Projects), and creating a new folder under the C:\Projects folder for each one of your new projects. Where you actually choose to keep the source code for your new projects is completely up to you, of course. However, for the purposes of the samples contained in the DataLayer.NET Users’ Manuals, we will be assuming that you are saving the projects under the folder C:\Projects.

## 3.2 SQL Server Northwind sample database required

All of the sample programs provided with the DataLayer.NET library use the sample Northwind database that gets installed with SQL Server (developer’s version or stand alone SQL Server). You will need access to an instance of the Northwind database in order to create and run the sample applications provided. You can determine whether you have a local instance of SQL Server running on your machine by looking at the tray icons on the right side of your toolbar. If SQL Server is running, you should see the following icon with a green arrow as follows:



If it is not present, then you probably do not have SQL Server running on your machine. If this is the case, then you have two options: You can either connect to another machine running SQL Server to access to the Northwind database, or you can go ahead and install the developer’s version of SQL Server onto your machine. If you want to install the developer’s version of SQL Server onto your machine, there is a CD-ROM labeled “SQL Server 2000 Developer Edition” that comes with Visual

Studio.NET than can be used to install the SQL Server. Also be sure to install the Service Pack 3a patches that are provided on a separate CD-ROM after you finish installing the SQL Server. The last thing you want is to be spending hours chasing down some weird error that has already been fixed by a Service Pack CD that has already been released.

### 3.3 High level steps to create the Quick Start Application

Here is a high level view of the steps required to create the Quick Start Application. You will be given specific instructions in the next section to perform these steps. This is presented here to give you an overview of the steps that are required:

- Create the new project using Visual Studio.NET.
- Add the Reference to the DataLayer.NET Assembly.
- Add a project level Imports declaration for the CDT.DATALAYER namespace.
- Use the Code Generator to generate a new class file for the PRODUCTS table.
- Import the new PRODUCTS class file into your project.
- Decorate the new PRODUCTS class with the appropriate Attributes for each column.
- Design a simple window with several controls for the user interface for the program.
- Code the **Connect** button to connect to the Northwind database.
- Code the **Retrieve** button to retrieve the rows into the DataGrid control.
- Test the *Connect* and *Retrieval* portions of the application.
- Add a **Total** button to demonstrate using the strongly typed interface.
- Add an **Update** button to demonstrate sending updates to the database.

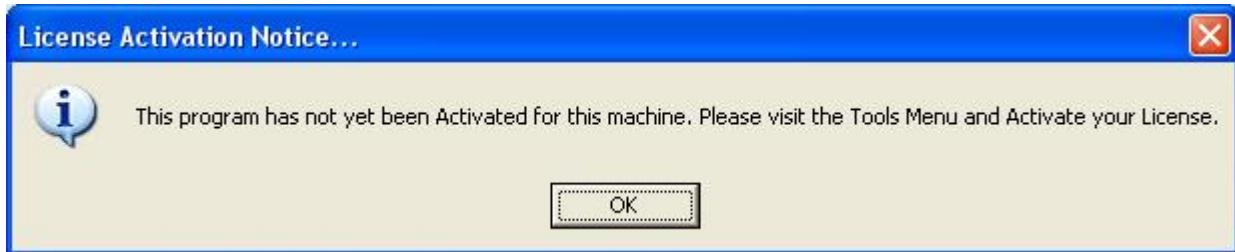
### 3.4 Steps to create the Quick Start Application

Use the following steps to create your first DataLayer.NET database application:

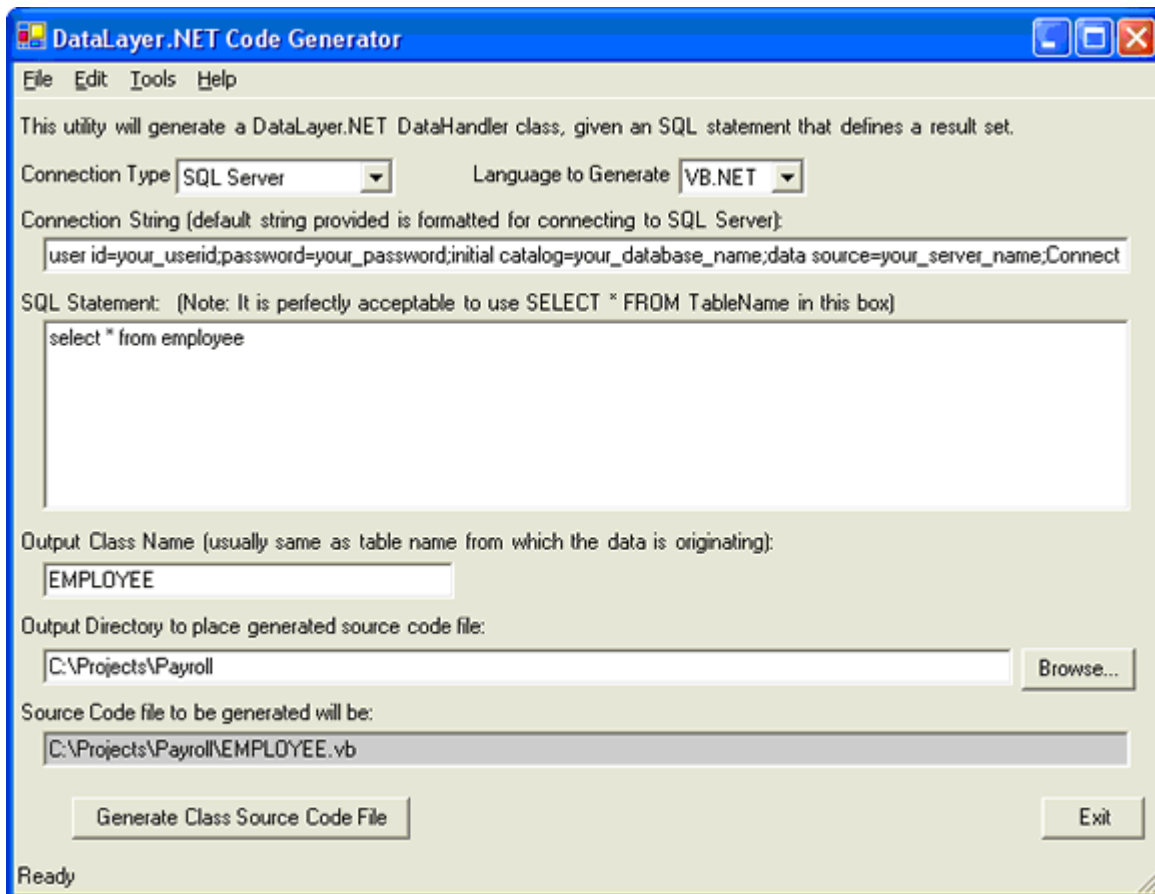
1. Create the C:\Projects folder, as mentioned above in section 3.1.
2. Start Visual Studio.NET, and click on the **New Project** link at the bottom.
3. Use the **Browse** button and select the folder "C:\Projects".
4. Make sure the "Visual Basic Projects" folder is selected on the top left box called "Project Types".
5. Make sure the "Windows Application" is selected in the upper right box called "Templates".
6. Type in "QuickStart" for the project's **Name**, and then click the OK button. You will get a simple windows application with one main form called "Form1.vb".
7. Expand the "References" folder in the Solution Explorer for the QuickStart application, and then right-click on the References and choose "Add Reference..." from the popup menu.
8. Click the **Browse** button and navigate to the C:\Program Files\DataLayer.NET folder, and select the DataLayer.dll file and click Open.
9. You will see the DataLayer.dll assembly selected in the bottom part of the window as follows:



17. Next, we will be using the DataLayer.NET Code Generator to generate a new class file to access the Products table in the Northwind database.
18. Minimize the Visual Studio.NET Environment, and launch the DataLayer.NET Code Generator using the desktop icon you created during the installation.  
**Note:** If you get the message box below, then refer to Appendix A at the end of the User's Guide document for more information about License Activation:



19. You should see the main Code Generator Program Window as follows:



20. For the Connection Type, you have options for a) SQL Server, b) ODBC, and c) OLE-DB databases. For this sample application, leave it as **SQL Server**.
21. For the Language to Generate, you have options for a) VB.NET and b) C#.NET. For this sample application, leave it as **VB.NET**.

22. For the Connection String, this will vary depending on the location (machine) where the Northwind database is located. Substitute the machine's network name (or IP Address) after the key words "data source=". Substitute the database name (i.e. – "Northwind" for this sample program) after the key word "initial catalog=". Finally, substitute your UserID and Password in the positions indicated by "user id=" and "password=".
23. For the SQL Statement, you can normally enter any SQL Statement that you would like that returns a result set. For this sample application, enter the following SQL Statement:  
select \* from products
24. For the Output Class Name, enter "PRODUCTS".  
NOTE: We usually create these particular class names in UPPER CASE. This will help you distinguish all the DataLayer.NET Code Generated entity classes in your application from all the other classes as you develop your application.
25. For the Output Directory, use the Browse button to select the folder C:\Projects\QuickStart.
26. If everything is set up correctly, you should notice the following complete path for the VB source code file that will be generated:  
C:\Projects\QuickStart\PRODUCTS.vb
27. Click the **Generate Class Source Code File** button. You should get the following confirmation window:



28. Click OK, and then exit the Code Generator Program using the **Exit** button.
29. Next, go back to the Visual Studio.NET environment so we can import the PRODUCT.vb source code file.
30. Right click on the QuickStart application and select "Add", and then "Add Existing Item..." from the sub-menu.
31. Select the PRODUCTS.vb source code file and click the **Open** button to import the file. Double-click on the PRODUCTS.vb source code file so you can see what was generated for you.

Here is a short sample of the file with some of the columns deleted:

```
Imports CDT.DATALAYER

' Class PRODUCTS generated by DataLayer.NET Code Generator.
<Serializable(> Public Class PRODUCTS
  Inherits DataHandler

  Sub New(ByVal DataConn As DataConnection, ByVal SQL As String)
    MyBase.New(DataConn, SQL)

    Me.UpdateTable = "PRODUCTS"
```

```

End Sub

Public Shared Function GetBaseSQL() As String
    Dim SQL As String
    SQL = "SELECT productid,productname,supplierid,categoryid,quantityperunit," & _
        "unitprice,unitsinstock,unitsonorder,reorderlevel,discontinued " & _
        "FROM PRODUCTS"
    Return SQL
End Function

<Updateable()> Public Property PRODUCTID(ByVal RowNum As Integer) As Integer
    Get
        PRODUCTID = GetIntegerData(RowNum, "PRODUCTID")
    End Get
    Set
        SetData(RowNum, "PRODUCTID", Value)
    End Set
End Property

<Updateable()> Public Property PRODUCTNAME(ByVal RowNum As Integer) As String
    Get
        PRODUCTNAME = GetStringData(RowNum, "PRODUCTNAME")
    End Get
    Set
        SetData(RowNum, "PRODUCTNAME", Value)
    End Set
End Property

<Updateable()> Public Property UNITPRICE(ByVal RowNum As Integer) As Decimal
    Get
        UNITPRICE = GetDecimalData(RowNum, "UNITPRICE")
    End Get
    Set
        SetData(RowNum, "UNITPRICE", Value)
    End Set
End Property

End Class

```

32. The general structure of these files is the Class name declaration at the top, followed by a **New** constructor and then a function called **GetBaseSQL**. This function is useful for building SQL statements in your descendant classes, as it provides all the columns in a bare SQL statement. After that, you have Get / Set properties defined for each column in the table. Notice that the data type and methods called by each property are particular to the datatype of each column.
33. Next you need to modify this class to decorate each column with the appropriate attributes. Here is a list of the available attributes:
- Updateable()** = The column is an updateable column (to be included in update statements sent to the database).
  - PrimaryKey()** = The column is part of the primary key for the table.
  - Identity()** = The column is an IDENTITY column (automatically generated number) in the database.
  - Sequencer()** = The column is a child table sequencer (don't worry about this for now, more

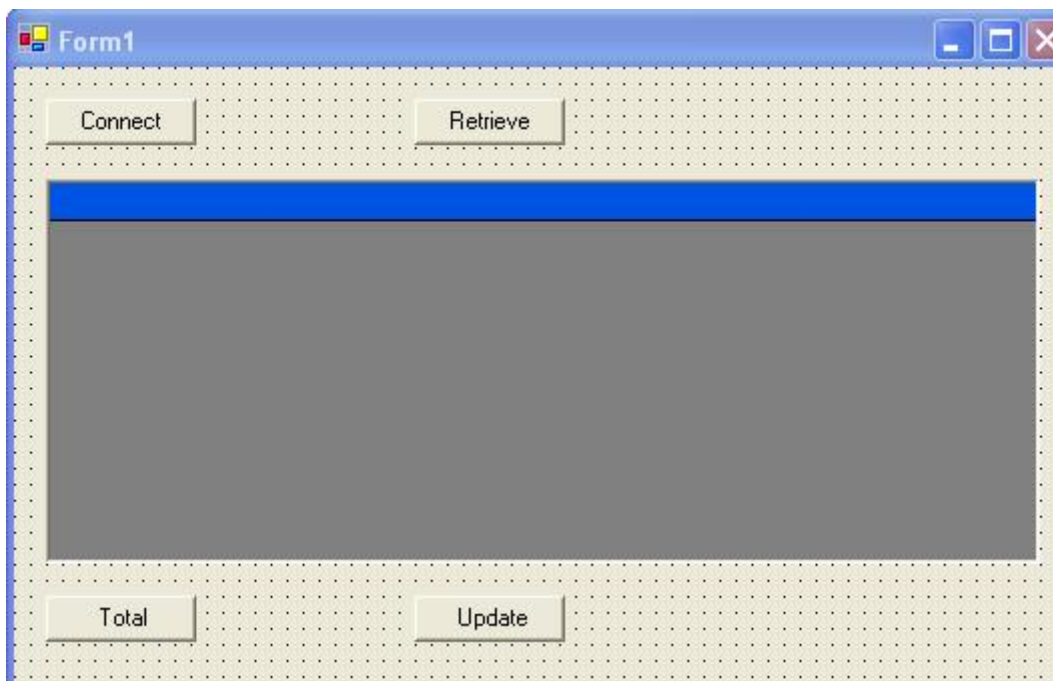
on this can be found in the User's Manual).

**DB2Timestamp()** = The column is a DB2 timestamp. DB2 requires a very specific format for sending updates and inserts based on these column types.

34. As you can see from looking at the generated source code, the "Updateable()" attribute is added by default by the code generator for every column. Note that when you are adding the attributes, you do not have to enter the parenthesis (they will automatically be added).
35. For the Products table in this example, the Primary Key is the PRODUCTID column. However, Microsoft chose to make this an auto-generating IDENTITY column (the value is automatically generated when a new Product is added to the database). Because of this, the PRODUCTID column is not an updateable column so you need to remove the Updateable() attribute and add the PrimaryKey() and Identity() attributes, as follows:

```
<PrimaryKey(), Identity(> Public Property PRODUCTID(ByVal RowNum As Integer) As _
Integer
Get
    PRODUCTID = GetIntegerData(RowNum, "PRODUCTID")
End Get
Set
    SetData(RowNum, "PRODUCTID", Value)
End Set
End Property
```

36. Click the "Save All" button to save your changes.
37. Next, let's work on creating the user interface for this sample application.
38. Click on the tab page containing the Form1.vb with the blank form visible on the screen.
39. Drag the lower right corner of the form to make it about twice as tall and twice as wide as it gets created originally.
40. Here is a print screen of the controls placed on the form, so you can see what it should look like as you place the controls on the form:



41. Using the toolbox on the left, add all four buttons and the DataGrid in the middle.
42. Set the .Text property of each button to read Connect, Retrieve, Total, and Update as shown above.
43. Set the .Name property to be btnConnect, btnRetrieve, btnTotal, and btnUpdate accordingly.
44. Leave the DataGrid control's name as "DataGrid1"
45. Click on the **View Code** button at the top of the Solution Explorer.
46. In the source code editor, enter the following two lines just below the section called "Windows Form Designer Generated Code":

```
Private mSQLConn As DataConnection ' DataLayer connection object.
Private mProducts As PRODUCTS ' DataHandler object for Products table.
```

The first line creates a module-wide variable called mSQLConn for the database connection. The second line creates a module-wide variable called mProducts that is a DataHandler object for working with the Products table.

47. Next, go back to the tab page containing the form designer for the Form1, and double-click on the Connect button to generate the btnConnect\_Click event.
48. Enter in the following lines of code for the btnConnect\_Click event:

```
' Create the DataLayer Connection object set for SQL Server mode...
mSQLConn = New DataConnection(DataLayer_ConnectionType.SQLServer)
mSQLConn.ConnectionString = "user id=guest;password=guestpass;" & _
    "initial catalog=Northwind;data source=localhost"

Try
    mSQLConn.Connect()
Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Critical, "Problem:")
Exit Sub
End Try
```

49. You will need to modify the line of code containing the ConnectionString property in the same manner as you did when you were using the Code Generator Program.
50. Next, go back to the tab page containing the form designer and double-click the Retrieve button to generate the btnRetrieve\_Click event.
51. Enter in the following source code for the btnRetrieve\_Click event:

```
' Create the PRODUCTS DataLayer object to do all the work for you...
mProducts = New PRODUCTS(mSQLConn, PRODUCTS.GetBaseSQL & " ORDER BY ProductName")

' Now retrieve the Product records from the database...
Try
    mProducts.GetAllRows()
Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Critical, "Problem:")
Exit Sub
End Try

' Now connect the DataSet to the DataGrid for displaying the records...
Dim dView As New DataView(mProducts.DataSet.Tables("data"))
DataGrid1.DataSource = dView
```

52. Notice above how we made use of the GetBaseSQL routine to generate most of the SQL Statement. We only needed to add the ORDER BY clause to complete our desired SQL Query.
53. Also notice in the 2<sup>nd</sup> to last line the mention of the DataSet table called "data". All DataHandler generated DataSet tables are called "data" by default. These DataHandler DataSet buffers usually only contain a single table object, so this is the standard table name that is used to access the row data.
54. Compile and run the sample application. When you run the program, first click on the **Connect** button (wait a couple of seconds to see if any error appears). Next, click on the **Retrieve** button to populate the Products row data into the DataGrid.  
Note: This display is very raw, as none of the columns have been formatted or sized accordingly, but it gives you an idea of the power the DataLayer.NET library puts into your hands without a lot of programming.
55. Next, we will be coding the **Total** button to display the total price of all the products in the database that have a price listed. This Total is not particularly useful information, but it will serve its purpose demonstrating the strongly typed interface.
56. Double-click the Total button to generate the btnTotal\_Click event skeleton.
57. Type the following code below into the event. Notice that when you are typing the mProducts object name, the IntelliSense feature of Visual Studio automatically gives you the column properties of the object in a list. All you have to do is type in the first letter or two of the property you are looking for, then hit the **tab key** to accept the selected property name.

```
' Add up all the product prices and display the total...
Dim dblTotal As Decimal = 0.0
Dim I As Integer

For I = 1 To mProducts.RowCount
    If (mProducts.UNITPRICE(I) <> Decimal.MinValue) Then
        dblTotal += mProducts.UNITPRICE(I)
    End If
Next

MsgBox("The Grand Total of all Unit Prices is " & Format(dblTotal, "$#,###,##0.00"))
```

58. Go ahead and compile and run the program, and test out the **Total** button. Don't forget that you need to first Connect and Retrieve before clicking the **Total** button, or you will receive an error. You should see a messagebox similar to the following (the exact total may vary):



59. Finally, let us code the **Update** button. You may be thinking that this will be the most complex coding of all, but the DataLayer.NET library takes care of all the work for you. It is essentially a single line of source code!
60. Double-click the **Update** button to generate the skeleton for the btnUpdate\_Click event.

61. Type the following code into the event:

```
' Save the changes the user made in the DataGridView back to the database...
Try
    mProducts.Update()
Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Critical, "Problem:")
Exit Sub
End Try
```

62. Compile and run the program. You will need to click the **Connect** and **Retrieve** buttons to fetch all the Products from the database. Next, make some changes to several product names (do not change the Product ID column values in the leftmost column – the database does not allow this, and you may get an error if you change the Supplier ID to a value that does not exist in the Supplier table, so you should probably stay away from that column as well). After you have made a few changes, click on the Update button to save your changes. To test whether your changes saved properly, you can click on the **Retrieve** button again to pull the records again from the database to see that your changes were indeed saved.

Note: A fully completed QuickStart application is included in the Zip file called QuickStart.ZIP.

### 3.5 Summary

In summary, you can already see from the QuickStart sample how much time and coding the DataLayer.NET library is going to be saving you. There are many more powerful features for you to learn about. Please take time to sit down and read the User's Manual for a more in depth look at the features and capabilities of the DataLayer.NET library.

### 3.6 Purchase Information

The cost of the DataLayer.NET™ Library is \$149.95 (US Dollars) plus 7.0% sales tax (applicable only for sales to Florida based customers). Because of the aggressive offering of this unique product at such a low price, we are only distributing our product electronically over the Internet to save on manufacturing and distribution costs. The 3-Volume set of manuals is delivered electronically in PDF format.

Purchase of this product includes one full year of free technical support. Additional annual support subscriptions (if needed) are available for only \$99.00 per year\*, which also entitles you to download and use the latest version of the library. Technical support is available via email submission of issues. On our website, we also have an on-line knowledgebase of solved customer cases that you can search to find solutions to common issues.

To purchase the DataLayer.NET™ Library, or to obtain more information, simply email us at [techsupport@creativedatatech.com](mailto:techsupport@creativedatatech.com), or call us at (850) 997-1464.

\* - Subject to future rate changes at the option of Creative Data Technologies, Inc.