**1**

**Introducing Visual C# and Databases**

**Preview**

In this first chapter, we will do a quick overview of what the course entails. We will discuss what you need to complete the course.

We’ll take a brief look at what databases are, where they are used, and how Visual C# is used with databases. And, we’ll review the Visual C# development environment and the steps followed to build an application in Visual C#.

**Course** **Objectives**

1. Understand the benefits of using Microsoft Visual C# to build a ‘front-end’ interface as a database programming tool
2. Learn database structure, terminology, and proper database design
3. Learn how to connect to a database using Visual C# data objects
4. Learn the use of Visual C# data bound controls
5. Learn to make database queries using SQL (structured query language)
6. Understand proper database search techniques
7. Learn how to ADOX (Active Data Object Extended) technology to create a database
8. Learn database management techniques
9. Learn to create and produce database reports
10. Learn how to distribute a Visual C# database application
11. Understand connection to different types of databases and remote databases
12. Introduce other database concepts

**Course Requirements**

An obvious requirement is a Windows-based computer with Microsoft Windows as well as Visual Studio 2019 Community Edition. The student should be familiar with the basics of using the Windows operating system.

No knowledge of databases or how to work with databases is presumed. Adequate introductory material is presented. Even if you’ve worked with databases before, it is suggested you read through this introductory information to become acquainted with the nomenclature used by the author for databases and their component parts.

This course does ***not*** teach you how to build a Visual C# application. It is assumed that the reader has a basic understanding of the Visual C# development environment and knows the steps involved in building a Visual C# application. You should feel quite comfortable with building the example application at the end of this first chapter. If not, our company, Kidware Software, offers a tutorial textbook that teach this information named “Learn C#”. Please visit our Visual C# web site at:

<https://www.kidwaresoftware.com/visual-c/>

**What is a Database?**

A **database** is a collection of **information**. This information is stored in a very structured manner. By exploiting this known structure, we can access and modify the information quickly and correctly.

In this information age, databases are everywhere:

1. When you go to the library and look up a book on their computer, you are accessing the library’s book **database**.
2. When you go on-line and purchase some product, you are accessing the web merchant’s product **database**.
3. Your friendly bank keeps all your financial records on their **database**. When you receive your monthly statement, the bank generates a **database** **report**.
4. When you call to make a doctor appointment, the receptionist looks into their **database** for available times.
5. When you go to your car dealer for repairs, the technician calls up your past work record on the garage **database**.
6. At the grocery store, when the checker scans each product, the price is found in the store’s **database**, where inventory control is also performed.
7. When you are watching a baseball game on television and the announcer tells you that “the batter is hitting .328 against left-handed pitchers whose mother was born in Kentucky on a Tuesday morning,” that useless information is pulled from the team’s **database** (apologies to our foreign readers who don’t understand the American game of baseball!).

You can surely think of many more places that databases enter your life. The idea is that they are everywhere. And, each database requires some way for a user to interact with the information within. Such interaction is performed by a **database** **management** **system** (**DBMS**).

The tasks of a **DBMS** are really quite simple. In concept, there are only a few things you can do with a database:

1. View the data
2. Find some data of interest
3. Modify the data
4. Add some data
5. Delete some data

There are many commercial database management systems that perform these tasks. Programs like Access (a Microsoft product) and Oracle are used world-wide. In this course, we look at using **Visual C#** as a **DBMS**.

Examples where you might use Visual C# as a DBMS:

1. Implementing a new application that requires management of a database
2. Connecting to an existing database
3. Interacting with a database via a server or the internet

In a DBMS, the database may be available **locally** on your (or the user’s) computer, available on a **LAN** (local area network) shared by multiple users, or only available on a **web** **server** via the Internet. In this course, we spend most of our time looking at local databases, but access with remote databases is addressed.

We will look at databases in more depth in the next chapter. You will see that databases have their own vocabulary. Now, let’s take a look at how Visual C# fits into the database management system.

**Where Does Visual C# Fit In?**

For database management, we say our Visual C# application acts as a **front**-**end** to the database. This means the Visual C# application provides the **interface** between the user and the database. This interface allows the user to tell the database what he or she needs and allows the database to respond to the request displaying the requested information in some manner.

A Visual C# application cannot directly interact with a database. There is a set of intermediate components between the application and the database known as **ADO** (ActiveX Data Object) **.NET** **data** **objects**:

User

Visual C# Application

Database

ADO .NET Data Objects

The **data** **objects** are Visual C# components that allow connection to the database, creation of data sets from the database and management of the database contents. These objects are the conduit between the application and the database, passing information back and forth between the two.

As mentioned earlier, there are many commercial products (Access, SQL Server, Oracle) that do database management tasks. You may be asking why use Visual C# as a database management system (DBMS) when these commercial products are available? There are two primary advantages to using Visual C# as a DBMS instead of Access:

1. Your users don’t need to have any commercial product installed on their computers or know how to use such products. This saves the users money.
2. By building a custom front-end, you limit what your user can do with the information within the database. Under normal operation, commercial DBMS provide no such limits.

So, in this course, we will look at how to build Visual C# applications that operate as front-ends to databases. Research has shown that over half of all Visual C# applications involve working with databases. We will look at how to make our applications into complete database management systems, being able to view, search, modify, add, and/or delete database information.

Before going any further, let’s review the steps in building a Visual C# application and then build a simple application for practice.

**Building a Visual C# Application**

In the remainder of this chapter, we will provide an overview of a Visual C# application and how the Visual C# development environment is used to develop an application. This should provide you with some idea of what knowledge you need to possess to proceed in this course and introduce the terminology used by the author to describe a Visual C# application.

**Structure of a Visual C# Windows Application**

Project

Control 1

Control 2

Form 3

Form 2

Form 1

Control 1

Control 1

Control 2

Control 2

Control 3

Control 3

Control 3

{Code}

{Code}

{Code}

**Application** (Project) is made up of:

* **Forms** - Windows that you create for user interface
* **Controls** - Graphical features drawn on forms to allow user interaction (text boxes, labels, scroll bars, buttons, etc.) (Forms and Controls are **objects**.)
* **Properties** - Every characteristic of a form or control is specified by a property. Example properties include names, captions, size, color, position, and contents. Visual C# applies default properties. You can change properties when designing the application or even when an application is executing.
* **Methods** - Built-in methods that can be invoked to impart some action to a particular control or object.
* **Event Methods** - **Code** related to some object or control. This is the code that is executed when a certain event occurs. In our applications, this code will be written in the C# language (covered in detail in Chapter 2 of these notes).
* **General Methods** - **Code** not related to objects. This code must be invoked or called in the application.

**Steps in Developing Application**

There are three primary steps involved in building a Visual C# application:

1. **Draw** the user **interface**
2. **Assign** **properties** to controls
3. **Write** **code** for event methods. Develop any needed general methods.

We’ll look at each step.

**Drawing the User Interface and Setting Properties**

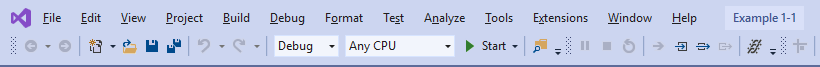
Visual C# operates in three modes.

1. **Design** mode - used to build application
2. **Running** mode - used to run the application
3. **Debugging** mode - application halted and debugger is available

We focus here on the **design** mode.

Several windows should appear when you start Visual C#. If any of these windows do not appear, they may be accessed using the main window menu **View** item.

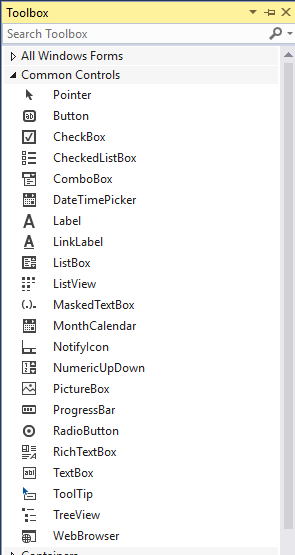
The **Main Window** consists of the title bar, menu bar, and toolbar. The title bar indicates the project name. The menu bar has drop-down menus from which you control the operation of the Visual C# environment. The toolbar has buttons that provide shortcuts to some of the menu options (ToolTips indicate their function).



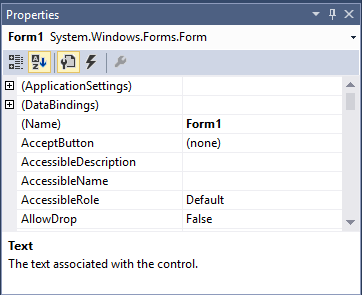
The **Form Window** is central to developing Visual C# applications. It is where you draw your application.



The **Toolbox** is the selection menu for controls (objects) used in your application.



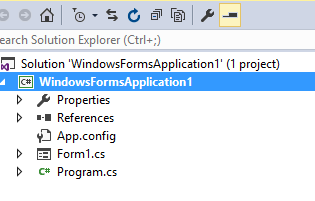
The **Properties Window** serves two purposes. Its primary purpose is to establish design mode (initial) property values for objects (controls). It can also be used to establish event methods for controls. Here, we just look at how to work with properties. To do this, click the **Properties** button in the task bar:



Properties button

The drop-down box at the top of the window lists all objects in the current form. Under this box are the available properties for the active (currently selected) object. Two property views are available: **Alphabetic** and **Categorized** (selection is made using menu bar under drop-down box). Help with any property can be obtained by highlighting the property of interest and pressing **<F1>**.

The **Solution Explorer Window** displays a list of all forms and other files making up your application

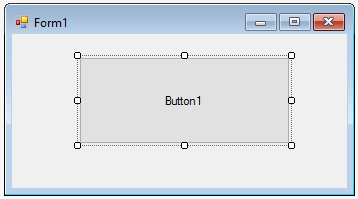


As mentioned, the user interface is ‘drawn’ in the form window. There are four ways to place controls on a form:

1. Click the tool in the toolbox and hold the mouse button down. Drag the selected tool over the form. When the cursor pointer is at the desired upper left corner, release the mouse button and the default size control will appear. This is the classic “drag and drop” operation.
2. Double-click the tool in the toolbox and it is created with a default size on the form. You can then move it or resize it.
3. Click the tool in the toolbox, then move the mouse pointer to the form window. The cursor changes to a crosshair. Place the crosshair at the upper left corner of where you want the control to be and click the left mouse button. The control will appear at the clicked point.
4. Click the tool in the toolbox, then move the mouse pointer to the form window. The cursor changes to a crosshair. Place the crosshair at the upper left corner of where you want the control to be, press the left mouse button and hold it down while dragging the cursor toward the lower right corner. A rectangle will be drawn. When you release the mouse button, the control is drawn in the rectangle.

To **move** a control you have drawn, click the object in the form (a cross with arrows will appear). Now, drag the control to the new location. Release the mouse button.

To **resize** a control, click the control so that it is selected (active) and sizing handles appear. Use these handles to resize the object.



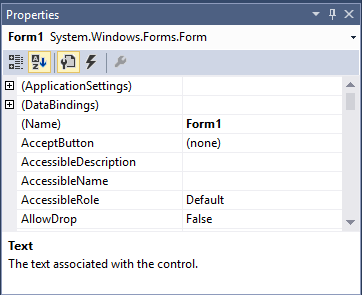
Use sizing handles to resize control

Click here to move object

To delete a control, select that control so it is active (sizing handles will appear). Then, press **<Delete>** on the keyboard. Or, right-click the control. A menu will appear. Choose the **Delete** option. You can change your mind immediately after deleting a control by choosing the **Undo** option under the **Edit** menu.

**Setting Properties of Controls at Design Time**

Each form and control has **properties** assigned to it by default when you start up a new project. There are two ways to display the properties of an object. The first way is to click on the object (form or control) in the form window. Sizing handles will appear on that control. When a control has sizing handles, we say it is the **active** control. Now, click on the Properties window or the Properties window button in the tool bar. The second way is to first click on the Properties window. Then, select the object from the drop-down box at the top of the Properties window. When you do this, the selected object (control) will now be active (have sizing handles). Shown is the Properties window (make sure the **Properties** button, not the **Events** button is selected in the toolbar) for the Form object:



Properties button

The drop-down box at the top of the Properties Window is the **Object** box. It displays the name of each object in the application as well as its type. This display shows the **Form** object. The **Properties** list is directly below this box. In this list, you can scroll through the list of properties for the selected object. You select a property by clicking on it. Properties can be changed by typing a new value or choosing from a list of predefined settings (available as a drop down list). Properties can be viewed in two ways: **Alphabetic** and **Categorized** (selected using the menu bar under the Object box). At the bottom of the Properties window is a short description of the selected property (a kind of dynamic help system).

A very important property for each control is its **Name**. The name is used by Visual C# to refer to a particular object or control in code. A convention has been established for naming Visual C# controls. This convention is to use a three letter (lower case) prefix (identifying the type of control) followed by a name you assign. A few of the prefixes are (we’ll see more as we progress in the notes):

**Control Prefix Example**

Form frm frmWatch

Button btn btnExit, btnStart

Label lbl lblStart, lblEnd

Text Box txt txtTime, txtName

Menu mnu mnuExit, mnuSave

Check box chk chkChoice

It is suggested to use a mixture of upper and lower case letters for better readability. But, be aware that Visual C# is a case-sensitive language, meaning the names **frmWatch** and **FRMWATCH** would not be the same name.

Control (object) names can be up to 40 characters long, must start with a letter, must contain only letters, numbers, and the underscore (\_) character. Names are used in setting properties at run-time and also in establishing method names for control events. Use meaningful names that help you (or another programmer) understand the type and purpose of the respective controls.

**Setting Properties at Run Time**

In addition to setting properties at design time, you can set or modify properties while your application is running. To do this, you must write some code. The code format is:

objectName.PropertyName = NewValue;

Such a format is referred to as dot notation. For example, to change the **BackColor** property of a button named **btnStart**, we'd type:

**btnStart.BackColor = Color.Blue;**

Good naming conventions make it easy to understand what’s going on here. The button named btnStart will now have a blue background.

**How Names are Used in Control Events**

The names you assign to controls are also used by Visual C# to set up a framework of event-driven methods for you to add code to. Hence, proper naming makes these methods easier to understand.

The format for each of these methods is:

**private** **void** ControlName\_Event(**Arguments**)

**{**

**}**

where **Arguments** provides information needed by the method to do its work.

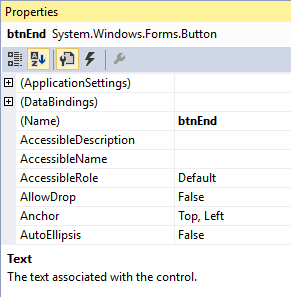
Visual C# provides the header line with its information and arguments and left and right curly braces (all code goes between these two braces) delineating the start and end of the method. Notice that with proper naming convention, it is easy to identify what tasks are associated with a particular event of a particular control.

**Writing Code**

The last step in building a Visual C# application is to write code using the **C#** language. This is the most time consuming task in any Visual C# application. It is also the most fun and most rewarding task. As controls are added to a form, Visual C# automatically builds a framework of all event methods. We simply add code to the event methods we want our application to respond to. And, if needed, we write general methods.

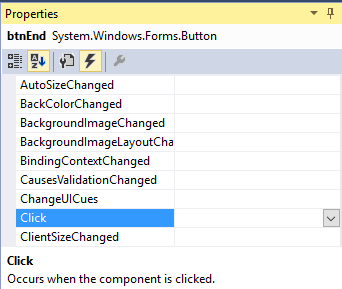
Code is placed in the **Code** **Window**. There are two ways to establish event methods for controls – one directly from the form and one using the properties window. Let’s look at both. Every control has a **default** **event** **method**. This is the method most often used by the control. For example, a button control’s default event method is the Click event, since this is most often what we do to a button. To access a control’s default event method, you simply double-click the control on the form.

Though simple and quick, double-clicking a control to establish a control event method will not work if you are not interested in the default event. In that case, you need to use the properties window. Recall we mentioned that the properties window is not only used to establish properties, but also event methods. To establish an event method using the properties window, click on the **Events** button (appears as a lightning bolt) in the properties window toolbar:



Events button

The active control’s events will be listed (the default event will be highlighted):



To establish an event method, scroll down the listed events and double-click the one of interest. The selected event method will appear in the code window.

There are a few rules to pay attention to as you type Visual C# code:

* Visual C# is case-sensitive, meaning upper and lower case letters are considered to be different characters. When typing code, make sure you use upper and lower case letters properly
* Visual C# ignores any “**white** **space**” such as blanks. We will often use white space to make our code more readable.
* Curly **braces** are used for grouping. They mark the beginning and end of programming sections. Make sure your Visual C# programs have an equal number of left and right braces. We call the section of code between matching braces a **block**.
* It is good coding practice to **indent** code within a block. This makes code easier to follow. The Visual C# environment automatically indents code in blocks for you.
* Every Visual C# statement will end with a semicolon. A **statement** is a program expression that generates some result. Note that not all Visual C# expressions are statements (for example, the line defining the form constructor has no semicolon).

**Review of Variables**

Variables are used by Visual C# to hold information needed by your application. Rules used in naming variables:

1. No more than 40 characters
2. They may include letters, numbers, and underscore (\_)
3. The first character must be a letter which, by convention, is usually lower case
4. You cannot use a reserved word (word needed by Visual C#)

**Visual C# Data Types**

1. **bool** (true or false)
2. **int**, **long** (Whole numbers)
3. **short, float**, **double** (Floating point numbers)
4. **DateTime**
5. **string** (Used for many control properties)
6. **char** (single character string variables)
7. **Object** (yes, objects can be variables!)

**Variable Declaration**

Once we have decided on a variable name and the type of variable, we must tell our Visual C# application what that name and type are. We say, we must **explicitly** **declare** the variable.

There are many advantages to **explicitly** typing variables. Primarily, we insure all computations are properly done, mistyped variable names are easily spotted, and Visual C# will take care of insuring consistency in variable names. Because of these advantages, and because it is good programming practice, we will always explicitly type variables.

To **explicitly** type a variable, you must first determine its **scope**. Scope identifies how widely disseminated we want the variable value to be. We will use three levels of scope:

* Block level
* Method level
* Form level

**Block** **level** variables are only usable within a single block of code (will be discussed in more detail in Class 2).

The value of **method** **level** variables are only available within a method. Such variables are declared within a method, using the variable type as a declarer:

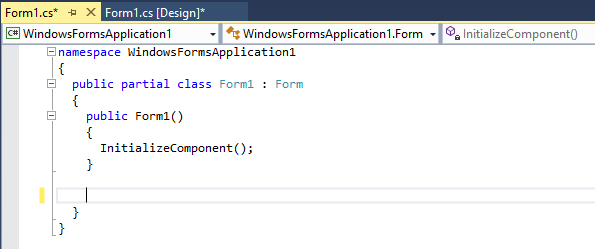
## int myInt;

## double myDouble;

**string myString, yourString;**

These declarations are usually placed after the opening left curly brace of a method.

**Form** **level** variables retain their value and are available to all methods within that form. Form level variables are declared in the code window right after the **Form constructor** generated automatically by Visual C#, outside of any other method:



Place form level variable declarations here, before any methods.

Form level variables are declared just like method level variables:

**int myInt;**

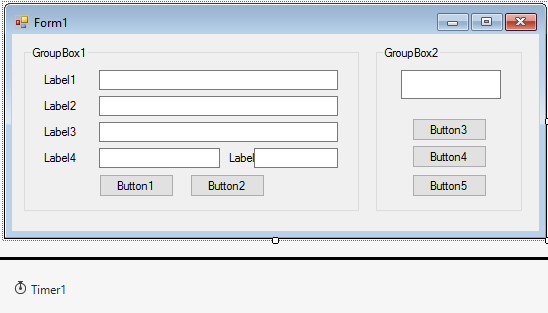
**DateTime myDate;**

**Example 1-1**

**Mailing List Application**

In this example, we will build a Visual C# application that could function as a database interface. The application allows the entry of information (names and addresses) to build a mailing list. An added feature is a timer that keeps track of the time spent entering addresses. After each entry, rather than write the information to a database (as we would normally do), the input information is simply displayed in a Visual C# message box. We present this example to illustrate the steps in building an application. If you feel comfortable building this application and understanding the corresponding code, you probably possess the Visual C# skills needed to proceed with this course.

1. Start a new project. Place two group boxes on the form (one for entry of address information and one for the timing function). In the first group box, place five labels, five text boxes, and two buttons. In the second group box, place a text box and three buttons. Add a timer control. Resize and position controls so your form resembles this:



1. Set properties for the form and controls (these are just suggestions – make any changes you might like):

**Form1**:

Name frmMailingList

FormBorderStyle Fixed Single

Text Mailing List Application

**groupBox1**:

Name grpMail

Text Address Information

Enabled False

**label1**:

Text Name

**label2**:

Text Address

**label3**:

Text City

**label4**:

Text State

**label5**:

Text Zip

**textBox1**:

Name txtName

TabIndex 0

**textBox2**:

Name txtAddress

TabIndex 1

**textBox3**:

Name txtCity

TabIndex 2

**textBox4**:

Name txtState

TabIndex 3

**textBox5**:

Name txtZip

TabIndex 4

**button1**:

Name btnAccept

Text &Accept

TabIndex 5

**button2**:

Name btnClear

Text &Clear

**groupBox2**:

Name grpTime

Text Elapsed Time

**textBox6**:

Name txtElapsedTime

Font Bold, Size 14

TabStop False

Text 00:00:00

TextAlign Center

**button3**:

Name btnStart

Text &Start

**button4**:

Name btnPause

Text &Pause

Enabled False

**button5**:

Name btnExit

Text E&xit

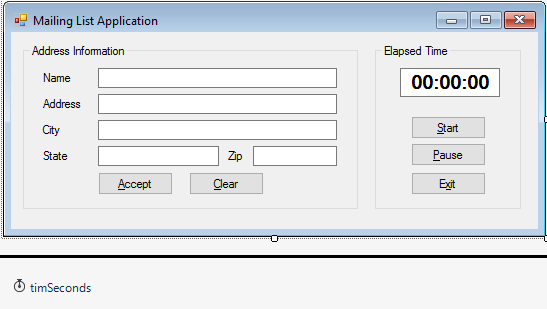
**timer1:**

Name timSeconds

Enabled False

Interval 1000

When done, the form should appear something like this:



1. Form level variable declarations:

**DateTime lastNow;**

**TimeSpan elapsedTime;**

1. Put this code in the **btnStart Click** event method:

**private void BtnStart\_Click(object sender, EventArgs e)**

**{**

**// Start button clicked**

**// Disable start and exit buttons**

**// Enabled pause button**

**btnStart.Enabled = false;**

**btnExit.Enabled = false;**

**btnPause.Enabled = true;**

**// Establish start time and start timer control**

**lastNow = DateTime.Now;**

**timSeconds.Enabled = true;**

**// Enable mailing list frame**

**grpMail.Enabled = true;**

**txtName.Focus();**

**}**

1. Put this code in the **btnPause Click** event method:

**private void BtnPause\_Click(object sender, EventArgs e)**

**{**

**// Pause button clicked**

**// Disable pause button**

**// Enabled start and exit buttons**

**btnPause.Enabled = false;**

**btnStart.Enabled = true;**

**btnExit.Enabled = true;**

**// Stop timer**

**timSeconds.Enabled = false;**

**// Disable editing frame**

**grpMail.Enabled = false;**

**}**

1. Put this code in the **btnExit Click** event method:

**private void BtnExit\_Click(object sender, EventArgs e)**

**{**

**this.Close();**

**}**

1. Put this code in the **timSeconds Timer** event method:

**private void TimSeconds\_Tick(object sender, EventArgs e)**

**{**

**// Compute elapsed time and display**

**elapsedTime += DateTime.Now - lastNow;**

**txtElapsedTime.Text = Convert.ToString(new TimeSpan(elapsedTime.Hours, elapsedTime.Minutes, elapsedTime.Seconds));**

**lastNow = DateTime.Now;**

**}**

Note a couple of lines in the code above are so long that the word processor wraps them around at the margin. Type each as one long line, not two separate lines. Be aware this happens quite often in these notes when actual code is being presented.

1. Put this code in the **txtInput KeyPress** event method (handles the KeyPress event for all input text boxes):

**private void TxtInput\_KeyPress(object sender, KeyPressEventArgs e)**

**{**

**String boxName = ((TextBox) sender).Name;**

**// Check for return key**

**if ((int) e.KeyChar == 13)**

**{**

**switch (boxName)**

**{**

**case "txtName":**

**txtAddress.Focus();**

**break;**

**case "txtAddress":**

**txtCity.Focus();**

**break;**

**case "txtCity":**

**txtState.Focus();**

**break;**

**case "txtState":**

**txtZip.Focus();**

**break;**

**case "txtZip":**

**btnAccept.Focus();**

**break;**

**}**

**}**

**// In zip text box, make sure only numbers or backspace pressed**

**if (boxName.Equals("txtZip"))**

**{**

**if ((e.KeyChar >= '0' && e.KeyChar <= '9') || (int) e.KeyChar == 8)**

**{**

**e.Handled = false;**

**}**

**else**

**{**

**e.Handled = true;**

**}**

**}**

**}**

1. Put this code in the **btnAccept Click** event method:

**private void btnAccept\_Click(object sender, EventArgs e)**

**{**

**string s;**

**// Accept button clicked - form label and output in message box**

**// Make sure each text box has entry**

**if (txtName.Text.Equals("") || txtAddress.Text.Equals("") || txtCity.Text.Equals("") || txtState.Text.Equals("") || txtZip.Text.Equals(""))**

**{**

**MessageBox.Show("Each box must have an entry!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Information);**

**txtName.Focus();**

**return;**

**}**

**s = txtName.Text + "\r\n" + txtAddress.Text + "\r\n";**

**s += txtCity.Text + ", " + txtState.Text + " " + txtZip.Text;**

**MessageBox.Show(s, "Mailing Label", MessageBoxButtons.OK);**

**btnClear.PerformClick();**

**}**

1. Put this code in the **btnClear Click** event method:

**private void BtnClear\_Click(object sender, EventArgs e)**

**{**

**txtName.Text = "";**

**txtAddress.Text = "";**

**txtCity.Text = "";**

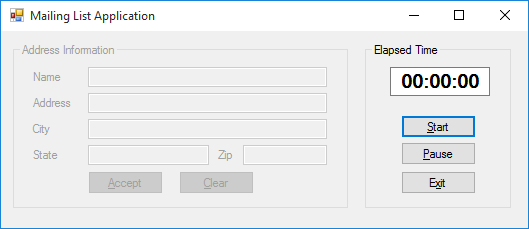
**txtState.Text = "";**

**txtZip.Text = "";**

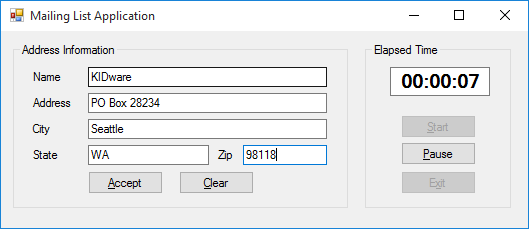
**txtName.Focus();**

**}**

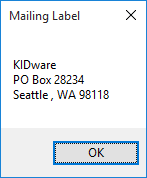
1. Save the application (saved in the **Example 1-1** folder in **VCSDB\Code\Class 1** folder). Run the application. Make sure it functions as designed. Here’s the running program.



Note that you cannot enter mailing list information unless the timer is running. Here’s the program after I entered some information:



and here’s what I see when I click **Accept**:



**Summary**

In this chapter, we introduced databases in general terms and how Visual C# can be used to develop a front-end application to interact with the database. And, we reviewed the steps involved in building a Visual C# application.

In the second chapter, we take a closer look at databases. We look at their structure, their terminology, and how they are constructed. You may be asking - when do we get to do some programming? The answer - in a couple more chapters. We want to make sure we have a firm foundation in place before diving into actual coding.