

# **Microsoft Visual Basic .NET 2005** *Walk-Through and Reference Guide*

Matthew Hagaman  
Consulting Software Developer and Web Designer

Ideal for creating:

- Non Graphics-Intensive Games
- Data-driven Applications
- Text/HTML/RTF Editors

## Table of Contents

<b>Introduction</b>	<b>4</b>
- Meet the Author	
- Pros and Cons of Visual Basic	
- How to Use This Book	
<b>Chapter 1: Getting Started</b>	<b>5</b>
- Installing Visual Basic 2005 Express Edition	
- The .NET Framework	
- The Start Page	
- Creating Your First Project	
- Toolbar Icons	
- Toolbox Components	
- Placing Components on the Form	
<b>Chapter 2: Layout</b>	<b>9</b>
- Layout Controls	
- Naming	
- Properties	
<b>Chapter 3: Laying Out the Word Processor</b>	<b>11</b>
- Why a Word Processor?	
- Creating and Manipulating the Form	
- Adding the Main Menu	
- Adding the RichTextBox	
<b>Chapter 4: Coding Basics</b>	<b>13</b>
- Ending a Program	
- A Look at Automatically Generated Code	
- Variables	
- Common Data Types	
- Commonly Used Operators	
- The Almighty Period, Parentheses, and Comma	
<b>Chapter 5: Changing the Properties of a Rich Text Box</b>	<b>16</b>
- Font	
- The If...Then Statement	
- The Message Box	
- Font Color	
- Page Color	
<b>Chapter 6: Input and Output using a Rich Text Box</b>	<b>19</b>
- Opening a File	
- Closing a File	
- Saving a File	
- The Difference Between Save and Save As...	
<b>Chapter 7: Manipulating Text</b>	<b>22</b>
- Using the Clipboard – Cut, Copy, & Paste	
- Select All	
- Finding Text	
- Printing	

<b>Chapter 8: Leftover Menu Items</b>	<b>24</b>
- Undo & Redo	
- Zooming	
- Running External Programs (Help)	
- Adding a Second Form (About this Program)	
- A Word About Version Numbers	
<b>Chapter 9: A Few Other Things</b>	<b>27</b>
- Other Events	
- MDI: Multiple Document Interface	
- Arrays	
- 0-based and 1-based Indexes	
- Modules	
- Structures	
- Do Loops	
- For...Next Loops	
- Substrings and Mid	
- Generating "Random" Numbers	
- Working with Timers	
- Focus	
<b>Chapter 10: Debugging</b>	<b>32</b>
- Breakpoints	
- Viewing the Value of Variables	
<b>Chapter 11: Working with GIMP</b>	<b>33</b>
- Where to Get GIMP	
- At First Glance	
- Drawing Tools	
- Using Filters	
- Saving and Exporting Pictures	
<b>Chapter 12: Distribution</b>	<b>35</b>
- Moving Your Visual Studio Project	
- Forms of Distribution	
- Install Creator	
- Patch Maker	
<b>Appendix 1: Glossary</b>	<b>36</b>
<b>Appendix 2: Recommended Software and References</b>	<b>39</b>

## Introduction

First of all, why should you buy this book? Well, you shouldn't. I typically don't charge for anything, be it writing or software development. I make money only by selling services, be that creating or modifying a program for business-specific needs.

## Meet the Author

Who am I? My name is Matthew Hagaman and I am a junior at Eastern Illinois University, an Elementary Education major, and a Computer Information Systems minor. While I do not claim to be a computer genius by any stretch of the imagination, I have been working with Visual Basic .NET for four years and am more than proficient enough to help others learn. Programs I have built have been primarily educational games and tools as well as some database and statistics-related programs I have created for my jobs. I also have a wealth of experience in the web design world, as I am currently the webmaster for six businesses and organizations.

But enough about me. My initial goal with this book is to provide a reference for all of my students. Since I am accustomed to using Visual Basic .NET 2002, this may also provide a reference to anyone who is making the switch from an earlier version of Visual Basic .NET. For more information about upgrading from Visual Basic .NET 2002 or 2003, see Appendix 3: Upgrading.

## Pros and Cons of Visual Basic

For starters, let me discuss the pros and cons of the language. Visual Basic (**VB**) is a great language for creating data-driven **applications** (particularly applications that reference Microsoft **Access Databases**), Text/**HTML**/RTF Editors, and non graphics-intensive games (board game style or old Nintendo-style RPG games), but IS NOT typically used for games with 3-dimensional graphics. A major drawback in using VB is that any programs you write can only be run on a computer running Windows 98 or later, and not on any alternative operating systems such as Linux or Macintosh OS.

## How to Use this Book

*This book is set up in a number of chapters, each progressively getting the reader acquainted with the language and its numerous components. Some text is formatted differently than the rest, for various reasons.*

Times New Roman is used to reference another part of the book.

**Bold text** draws attention to the first time a technical word or term is used.

`Courier New` is used to indicate actual VB code.

“Quotes” are used to point to text you should see on the screen.

**Red Text** indicates the text discussing changes in .NET 2005 from earlier versions.

Also, at the back of the book you will find a number of important appendices. Appendix 1 is a comprehensive glossary (which may come in handy when reading, if you're unfamiliar with a particular term or an acronym). Appendix 2 is a list of recommended programs (most of which are free downloads) as well as great written references.

I hope you are able to use this book as a stepping stone as you learn Visual Basic and as a reference after you are already familiar with the language. If you find you need help, feel free to send an e-mail to [mthagaman@gmail.com](mailto:mthagaman@gmail.com) and I will try to help you as much as I can.

## Chapter 1: Getting Started

- *Installing Visual Basic 2005 Express Edition*
- *The .NET Framework*
- *The Start Page*
- *Creating Your First Project*
- *Toolbar Icons*
- *Toolbox Components*
- *Placing Components on the Form*

### Installing Visual Basic 2005 Express Edition

For the first time, Microsoft is offering an Express version of **Visual Studio**<sup>1</sup> for free, and may be downloaded at least through the second half of 2007<sup>2</sup>. Express versions are available for each component of Visual Studio and have their own documentation packages. At last check, the programs are available for download (in their complete, registration-free package) at <http://msdn.microsoft.com/vstudio/express/support/install/>. If you need to check on the latest system requirements, visit <http://msdn.microsoft.com/vstudio/express/support/readme/>.

The installation is fairly self-explanatory, the only real choices come when deciding which components to install. Depending on your planned use of Visual Basic (whether you will need to access large quantities of data, particularly **Microsoft JET/ Access Databases**), you may or may not want to install **SQL Server 2005 Express**. If you think you may need to access data, you should install it, it never hurts to be prepared. Everyone should install the MSDN Express Library, which provides (somewhat limited) documentation which will come in handy when you run into a problem. The other two packages, .NET Framework 2.0 and Visual Basic .NET 2005 Express Edition, are required.

If you choose to purchase a more advanced version of Visual Studio, the installation may differ from above. More information on different versions of Visual Studio is available at <http://msdn.microsoft.com/vstudio/products/compare/>. I am currently using the Standard edition which I received through a Microsoft promotion.

### The .NET Framework

Before Visual Basic .NET was released and developers were largely using Visual Basic 6.0 to develop Windows applications, components required for the program to run had to be manually copied or installed on the user's computer. This was a somewhat frustrating process for developers because they often overlooked a component and their users were unable to run the program. With the release of Visual Studio .NET, Microsoft switched **distribution** models to require that developers not only distribute their application, but the entire **.NET Framework** package as well. This has been of great assistance to many developers, but greatly increases the size of any distribution.

### The Start Page

---


1 Visual Studio: The name given to the software that helps developers create **ASP**, C++, C# (pronounced C sharp), J#, and VB programs. While you may only have one language installed, I will refer to the Visual Basic **IDE** as Visual Studio from now on.

2 The Second Half of 2007: When the next version of Visual Studio will be released and the decision will be made to determine if Visual Studio 2007 will have a free/Express version.

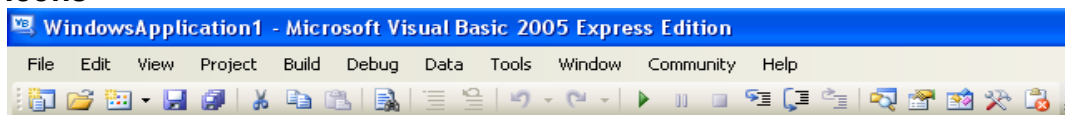
The first screen you see after having installed Visual Basic 2005 (and, if you're running it for the first time, letting the program configure itself) is the Start Page. The Start Page provides quick access to recently opened projects, creating projects, and developer news<sup>3</sup>. It also provides links to download sample applications and connect with other developers.

## Creating Your First Project

To create a project in Visual Studio, either click “Project...” following “Create:” on the start page or go to File > New Project. You will see that you can create a variety of projects, but the primary project type we will use is “Windows Application”. Create a Windows Application now, and call it anything you wish. I would suggest leaving the name “WindowsApplication1”, a name we can use to indicate we can delete the project later.

Try to acquaint yourself with the new environment, paying close attention to the menu, toolbar, “Toolbox” on the left, “Solution Explorer” on the right, and “Properties” in the lower right. While you are investigating, it may be useful to know that the  pushpin icon will change the element from auto-hiding to always showing or vice versa. I typically have everything but the properties window autohide.

## Toolbar Icons



While some of this may not make sense right away, the toolbar **icons** from left to right are:

**New Project:** Create a new project to build a new program.

**Open File or Project:** Open a File (including HTML files, icons, and other, more obscure, formats) or Project.

**Add New Item:** Add a new item such as a new Windows Form, Module, etc. These elements will be discussed later.

**Save:** Save the item you are currently editing. (I rarely use the “Save” button, I typically only use “Save All”)

**Save All:** Save everything associated with your project.

**Cut, Copy, & Paste:** Copy and Remove, Copy, or Replace an item from the clipboard<sup>4</sup> into your project.

**Find:** Locate a word or phrase (be it English or Visual Basic) within the page you currently have open.

**Comment Out (Shown Disabled)**<sup>5</sup>: When used in Code view, this makes the program overlook the selected code without deleting it.

**Uncomment (Shown Disabled):** When used in Code view, this makes the program recognize and use the selected code.

**Undo & Redo (Shown Disabled):** Take back the last action you performed, or do it again.

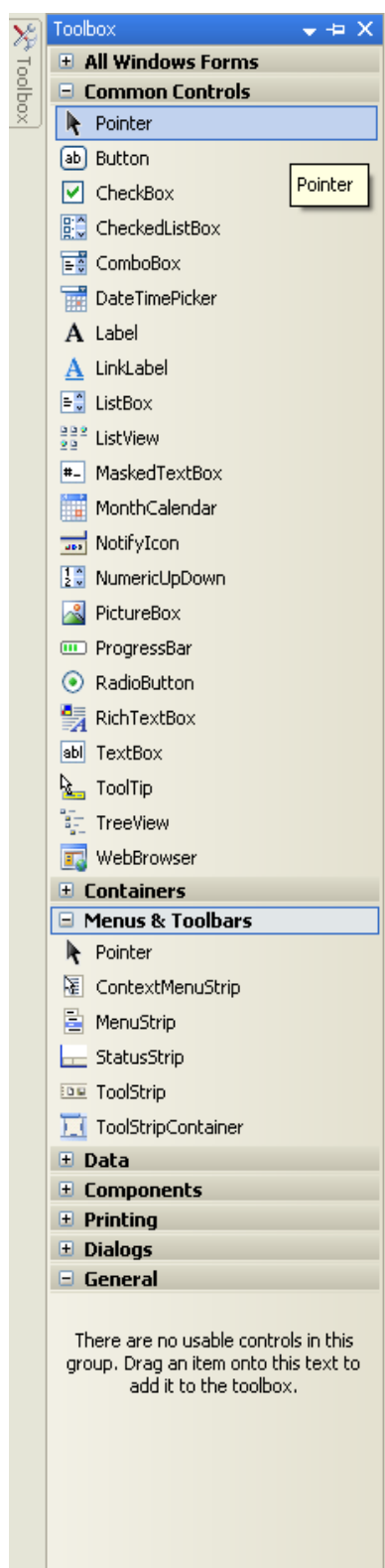
**Start Debugging**<sup>6</sup>: Run your program as the user will see it and pause the program when errors occur.

<sup>3</sup> Developer News: News headlines of interest to developers which is pulled from the Internet if you are connected when the program is first opened.

<sup>4</sup> Clipboard: The Windows Clipboard can be used to transfer text, images, or design components from one place within the application to another or between applications.

<sup>5</sup> Disabled: Disabled simply means the button is ghosted or grayed out and may not currently be used, in this case because we are in design view and not code view.

<sup>6</sup> Debugging (Debugger): When you choose start debugging, Visual Studio will run your program as is and allow you to use it as the user will. When an error of any kind occurs, Visual Studio will report it to you along with suggestions of how to fix it.



**Break/Pause Debugging (Shown Disabled):** Pause the program as it runs, and select the code that is currently being processed.

**Stop Debugging (Shown Disabled):** End the program and debugging to make changes.

**Step Into:** Run the next line of code.

**Step Over:** Run the next line of code after the current statement.

**Step Out:** Re-run the code from the beginning of the current statement.

**Solution Explorer, Properties Window, Object Explorer, Toolbox, & Error List:** Show the respective windows (we will explore these later).

## Toolbox Components

The toolbox is the repository for all **controls** that are used in a program. While it is possible for a developer to create their own controls, it is rare because Visual Studio provides so many to begin with. The “Common Controls” are the controls you will use most. I will describe the use of each.

**Pointer:** If you change your mind and decide you do not want to add a new control, choose the pointer to resume normal editing.

**Button:** The control users are most drawn to...they know something will happen when they click it. Typically used to start or stop an action.

**CheckBox:** The best way to have a user enter a true/false answer. Example: Receive junk e-mail?

**CheckedListBox:** A step above the list box, this is used to allow the user to enter a true/false answer to a bunch of questions at once. Example: Which items do you want to buy?

**ComboBox:** A great way to present a list in a small amount of space. Clicking a downward-pointing arrow will present the user with a list of choices.

**DateTimePicker:** A great way to present a calendar in a small amount of space. Clicking the downward-pointing arrow will cause a calendar to appear on-screen.

**Label:** Perhaps the most used control available, the label is used to tell the user what they should do with a control. Example: A label reading “Items To Print” above a CheckedListBox tells the user that any items they check will be printed.

**LinkLabel:** The LinkLabel is typically used to point the user to a web page - not a control I commonly use.

**ListBox:** Presents the user with a list of items to select from.

**ListView:** Think files...large icons with text beneath that can be used to open files, link the user to a web page, open another component of the program, etcetera.

**MaskedTextBox:** The masked text box, **new in Visual Basic .NET 2005**, allows the programmer to specify a format for data being entered. Example: ###-#### for a phone number.

**MonthCalendar:** Allows the user to choose a date from a calendar.

**NotifyIcon<sup>7</sup>:** An icon that will appear not as part of the program, but in the user's computer's system tray - in the bottom right-hand corner of the screen, next to the clock.

<sup>7</sup> Like many other controls, the NotifyIcon cannot be added to a form, but instead appears in a tray of sorts at the bottom of Visual Studio. It will not appear on the form, but may appear elsewhere on the user's screen, in this case as an icon in the system tray of the task bar (on the bottom right-hand corner of the screen, next to the clock).

**NumericUpDown:** Allows the user to choose a number from a list; a simple version of the ListBox.

**PictureBox:** Places an image or icon on the screen.

**ProgressBar:** Gives the user a visual indication of how far a process has progressed (seen most often in **software** installations).

**RadioButton:** Allows the user to choose only one option, as opposed to the CheckBox, which lets the user choose any or all options.

**RichTextBox:** An advanced version of the TextBox, the RichTextBox allows the user to open, edit, and save not only **Plain Text**, but **Rich Text** (RTF) as well.

**TextBox:** Typically used to allow the user to view or enter a smaller amount of text.

**ToolTip:** A control that allows the developer to provide the user with information when they hold the mouse over an object. The boxed “Pointer” on the previous page is an example of how ToolTips appear to the user.

**TreeView:** Gives the user a hierarchical view of items, or a view that indicates rank or progression. The classic computer example is where the upper level would be a folder, and within it are files.

**WebBrowser:** A very rich control that required more complex references in previous versions of Visual Studio .NET, this allows the user to browse the web (and any files that can be viewed in a web browser) from within your application.

Other controls that are important, but not included in the “Common Controls” list are:

**ContextMenuStrip:** Provides the user with a list of options when they right-click. Examples include (when right-clicking a RichTextBox) cut, copy, paste, font, color, and print.

**ImageList:** Primarily used for icons within toolbars, the ImageList makes it easy to build many images into the program and make them easily accessible to the developer.

**MenuStrip:** The standard File, Edit, etc. menu bar that can be customized with any heading, item, or hierarchy.

**StatusStrip:** Used to give the user constant access to changing information. For example, an item in the status bar of a word processing program may be “[current page] out of [total pages]”.

**Timer:** As you might imagine, the timer is used to cause a delay between one action and another.

**ToolStrip:** The ToolStrip control is the standard toolbar that appears in many programs, with customizable icons, text, button types, etc.

**TrackBar:** The trackbar is a control that is like a virtual knob, typically used to allow subtle variations in value, such as subtle changes in volume.

In addition, all of the printing and **dialog** controls are very important, but will be investigated as we move on. Other controls involved in layout will be discussed in Chapter 2: Layout.

As you may have noticed, each control's name does not contain any spaces. This will be true throughout your development experience; anything you create and name may not have spaces in the name.

## Placing Components on the Form

Most of the components we have discussed can be dragged from the toolbox onto the form or copied by first clicking the name of the component in the toolbox and then clicking the desired location on the form. The **Form** is another name for screen real estate, or the space that you have to place items on. The Form will appear virtually the same way in the application program as it does on the development screen. You may drag items around and resize them as you see fit. Note that not all items will not appear on the form, as some will appear only when called upon on, particularly the dialog boxes. For more information on placing components on the screen, see Chapter 2, where we explore the layout of a Windows Form.



## Chapter 2: Layout

- *Layout Controls*
- *Naming*
- *Properties*

### Layout Controls

There is a much wider variety of layout controls available in Visual Studio .NET 2005 than there was in earlier versions. The new layout tools alone are a good reason to make the switch.

**GroupBox:** A group box is simply a box with an engraved-looking line that acts as a form on top of a form. It is a small piece of real estate that groups all controls within it so when the GroupBox is moved, all of the controls in it move at the same time.

**Panel:** A control that, like the GroupBox, acts as a form on top of a form; without the engraved line.

**TabControl:** This control allows you to pack more onto a form. Clicking a tab will change the visible 'page.'

**TableLayoutPanel:** An easy way to organize controls in columns and rows.

**ToolStripContainer:** A toolbar control that contains buttons of either icons or text.

In addition to new controls, Visual Studio 2005 includes a number of new tools that allow the designer to easily space and center controls within the form. These tools are available in the "Format" menu.

### Naming

Everything in Visual Basic must be named. When you place a control on the form, its name is automatically assigned as ControlType# (such as TextBox1). When you need to do anything in Visual Basic code, it makes things much easier to have controls well-labeled, so when I place any control on the form, I name it (which is achieved by changing the "Name" property in the Properties window. When naming items, I like to use a formula to enable easy identification of both the type of control and its purpose. You can, of course, adopt your own method, but in any examples I give, I will use this naming system. The prefixes that designate the type of control are as follows.

**Button:** btn

**CheckBox:** chk

**ComboBox:** cmbo

**ListBox:** lst

**Label:** lbl

**PictureBox:** pic

**ProgressBar:** prg

**MenuItem:** mnu (may no longer be applicable in VB2005, as discussed in the menu section)

**RadioButton:** rdo

**RichTextBox:** rtb

**TextBox:** txt

Items not used frequently do not need to be named. When using a single timer or dialog (SaveFileDialog, PrintSetupDialog, etc), it's easy to remember a name like Timer1.

Also, it is not necessary to name items that will never have an action assigned to them, such as labels, top menu items (File, Edit, View, etc.), or, in some cases, group boxes.

**Properties**

Nearly everything done in Visual Basic is done by altering properties of the controls that have been placed on the form. Properties can be changed in one of two ways: by using the properties window during design (as explored earlier) or by altering the properties directly in the code. Examples of properties that are important in layout are the BackColor, ForeColor, BorderStyle, Font, etcetera.

We'll explore properties further in our main project, a word processor.

## Chapter 3: Laying Out the Word Processor

- *Why a Word Processor?*
- *Creating and Manipulating the Form*
- *Adding the Main Menu*
- *Adding the RichTextBox*

### Why a Word Processor?

While a word processor may seem to be a very simple program, it can actually be quite complex, and with it we can cover nearly everything used in VB .NET programs.

### Creating and Manipulating the Form

To begin, open Visual Studio 2005 and create a new project. Make sure "Windows Application" in the right pane is selected (if you do not see it, select Visual Basic > Windows in the left pane) and name it "WordProcessor1".

In the properties window on the right side of the screen, change "Size" to "1024, 768". The majority of monitors (somewhere around 70%) in use throughout the US and the world are set to a 1024x768 pixel resolution, so this is a good default for most forms in most programs.

While we're at it, set "StartPosition" to CenterScreen, and disable the option to maximize the window (set "Maximize Box" to False). Also, set "Text" to "Word Processor". You will notice that after changing the value of "Text", the new value appears in the top left-hand corner of the form.

### Adding the Main Menu

Now we will begin to place various controls on the form. To begin, place a MenuStrip (under Menus & Toolbars in the toolbox) on the form and fill in the following text / commands. The underlined items should be at the very top of the menu structure, with each of the other items underneath them.

<u>&amp;File</u>	<u>&amp;Edit</u>	<u>&amp;View</u>	<u>&amp;Help</u>
&Open...	&Undo	&Font...	&Help...
&Close	&Redo	Font &Color...	-
-	-	-	&About this Program
&Save	Cu&t	&Page Color	
Save &As...	&Copy	-	
-	&Paste	&Zoom	
&Print...	-		
-	Select &All		
E&xit	-		
	&Find...		
	Find &Next		

When entering these items, you will notice that the dashes create separators, or engraved lines between items. This is simply to group similar objects. Also, you will quickly notice the "&"s in the item text does not show up when you move on to create the new item. Instead, the character (letter) that follows the "&" is underlined. This indicates the user can use the Alt menu shortcuts, meaning the user can press Alt and then two or three keys instead of navigating the menu with the mouse. For example, in File > Exit, if you entered it

as "&File" and "E&xit", the user could press Alt, F, then X to exit. Note that the & precedes the "most significant character" and can be any letter not repeated in the same menu structure. For example, the significant letter in "Cut" is the "t" because "C" is used by in the Edit menu by "Copy" and "U" is used by "Undo".

Similarly, you can designate keyboard shortcuts for users to use as well. Following are a few common keyboard shortcuts used in nearly every program. If you allow keyboard shortcuts in your program, and use these commands, it is best to use these shortcuts so users do not have to learn a completely new set of shortcuts for your program.

Open - Ctrl + O  
Save - Ctrl + S  
Print - Ctrl + P  
Exit - Alt + F4  
Undo - Ctrl + Z  
Redo - Ctrl + Y

Cut - Ctrl + X  
Copy - Ctrl + C  
Paste - Ctrl + V  
Select All - Ctrl + A  
Find - Ctrl + F  
Help - F1

### Adding the RichTextBox

Find the RichTextBox in the toolbox (under Common Controls) and place it on your form. Before resizing it, make sure the Form itself is still sized "1024, 768" (sometimes adding a menu will change the size of the form). Then, stretch the RichTextBox so it fills the remainder of the screen. As you move or resize the box, the IDE will help you keep the size from going past the border of the form. As you drag, you should notice blue lines at the edge of the form. **This is a new feature in Visual Studio 2005**, and it is very useful in spacing every control on your form a standard distance from each other and the borders. Similarly, if you are using multiple TextBoxes, a purple line aligns the text itself instead of the TextBoxes' edges.

## Chapter 4: Coding Basics

- *Ending a Program*
- *A Look at Automatically Generated Code*
- *Variables*
- *Common Data Types*
- *The Almighty Period, Parentheses, and Comma*

### Ending a Program

In this chapter we are going to begin working with Visual Basic code, which means we have to switch to code view. The easiest way to switch from design view to code view is by double-clicking the control you will use to initiate an action. In this case, we are going to double-click the File > Exit menu item. In the space that appears, we are going to type `End`. When the user selects Exit from the File menu, the program will end.

### A Look at Automatically Generated Code

When you wrote the code to exit the program, you probably noticed the code that was already on the screen.

```
Public Class Form1

    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click
        End
    End Sub
End Class
```

At the top of it all are the words "Public Class", in blue, and "Form1", in black (unless, of course, you renamed the form). "Public" simply means the contents of the form are accessible to any part of your program. Similarly, "Private" indicates that the contents of the sub are only accessible to the Sub itself. A "Sub" is called a number of things, including procedure or function. It is simply a segment of the code, in this case called to action when the user clicks the ExitToolStripMenuItem (as indicated by "Handles ExitToolStripMenuItem.Click"). While "ExitToolStripMenuItem\_Click" may look important, it is not actually part of the code beyond being the name of the sub. You can change the name, though it is recommended that you leave it as it is automatically named, because it is a very descriptive name. "sender" and "e" are both **variables** that can be used within the sub.

### Variables

Understanding **variables** is very important in any language, but particularly in Visual Basic. When a variable is created, a small space is set aside in your computer's memory. This space will hold anything you put into it (such as text or numbers). The type of data the variable holds is usually declared when the variable is created.

```
Dim HP as integer = 100
```

In the above example, the variable HP (short for Health Points in many games) is created (or dimensioned) to hold a specific data type: a number - more specifically an **integer**, or a whole number. At the same time HP was dimensioned, its value was set to 100.

Variables are very important in any program, but it is important to note that variables are saved into the computer's memory and not the hard drive. When your program is closed (or sooner, depending on where the variable is declared, see the next paragraph) the variable and its value will be deleted, so other programs have access to the memory space that the variable was using. A variable (or data) can be saved in a non-volatile manner, and these procedures will be discussed in Chapter 6.

You have already seen three ways a variable can be created in Visual Basic code (Public, Private, Dim). Public variables (or procedures) can be accessed anywhere within a form (or anywhere in your program if created in a module - to be discussed later). Private variables (or procedures) can only be accessed by lower hierarchy code (indicated by the tabbed spaces at the beginning of every line of code). Dimmed variables can be accessed by any code at the same hierarchy or any code at a lower hierarchy. There are two additional ways to create a variable, but they can only be used within the parentheses of a procedure. "ByVal" variables (see an example in A Look at Automatically Generated Code above) and "ByRef" variables will be discussed later.

### Common Variable Types

**Boolean** - A boolean variable can contain one of two values: true or false. It takes up almost no space in memory and is ideal to use if you need to store information with only two possibilities, for example true/false, yes/no, right/wrong, dead/alive...

**Char** - A char variable will store a single character, be it letter, number, or punctuation mark.

**Date** - A date variable will store a day and time of day. While there are many formatting options, one valid date would be 6/11/2007, 13:00.

**Decimal** - A decimal variable is used when a precise number needs to be saved. If only a whole number is needed, Integer should be used because it uses less memory.

**Integer** - A integer variable will store any whole number.

**Stack** - A variable that stores and recalls data in the order in which it is added. An excellent example of use for this type of variable is in a web browser. When you click any link, the destination is added to the top of the 'Back' stack. When you click 'Back', you take the last destination from the top of the stack, load it, and add it to the top of the 'Forward' stack. When you click 'Forward', you take the page at the top of the 'Forward' stack, load it, and add the page's **URL** to the top of the 'Back' stack.

**String** - A string variable will store a string of characters, or any text, be it a word, phrase, sentence, or paragraph. *A string is always surrounded by quotation marks.*

### Commonly Used Operators

**=** The equal sign is used whenever you want to set a property or variable to a value. For example, `RichTextBox1.Text = "bologna"` would set the text in RichTextBox1 to "bologna".

**+, -, \*, /** These symbols are used as you might expect, mathematically. + is used to add numbers, - to subtract, \* to multiply, and / to divide.

**&** The ampersand symbol is used to combine strings. For example, `RichTextBox1.Text = "Bologna" & " and " & "cheese"` would set the text in RichTextBox1 to "Bologna and cheese".

**+=** When dealing with numerical variables, it is often useful to use the += shortcut. Saying `Count += 1` is the same as saying `Count = Count + 1`.

**-=** Similarly, saying `Count -= 1` is the same as saying `Count = Count - 1`.

### **The Almighty Period, Parentheses, and Comma**

Intriguing Section Heading? It should be. The three most used characters in Visual Basic code are the period, the parentheses, and the comma. As you will see in Chapter 5, the color of the background of a `RichTextBox` is accomplished by using code similar to `RichTextBox1.BackColor = Color.Black`. As you can see, the period allows you to change properties when entered after an object's name. While the period gives you access to an object's properties, parentheses give you access to an object's functions. For example, as you will see in Chapter 7, a `RichTextBox` can be searched using code like `RichTextBox1.Find("Text to find", RichTextBoxFinds.MatchCase)`. The comma, as in the last example, allows you to specify different options within a function.

## Chapter 5: Changing the Properties of a Rich Text Box

- *Font*
- *The If...Then Statement*
- *The Message Box*
- *Font Color*
- *Page Color*

I'm sure by now you hate me. Or Visual Basic. Or both. Well, chin up! Now we get into learning by doing. Make sure your WordProcessor1 project is still open and read on!

### Font

In the code view mode, click on the first dropdown under the tabs. Select "FontToolStripMenuItem" and then select "Click" from the dropdown on the right. This is an alternate way of automatically generating the control code (an alternative of double-clicking the item in the design view). Type the following:

```
Dim FontDialog1 As New FontDialog
FontDialog1.Font = RichTextBox1.SelectionFont
If FontDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
    RichTextBox1.SelectionFont = FontDialog1.Font
End If
```

The first line of code does the same thing dragging a FontDialog from the toolbox onto the form in design view would do. We created it in the code so it would be more memory-efficient. This can be done with any control, though the control will be eliminated (or disposed of) at the end of the Sub. The second line of code makes sure the font options already selected in the dialog when it opens are the same as any selected text in RichTextBox1. The FontDialog1.ShowDialog() code brings the dialog onto the screen so the user can interact with it. The third line simply applies the font selections the user made in the dialog to any selected text in RichTextBox1. If you want the changes to be applied to all the text in RichTextBox1, replace RichTextBox1.SelectionFont with RichTextBox1.Font.

### The If...Then Statement

The If...Then statement will likely be used hundreds of times in any program. It allows you to specify code to run only under certain conditions. In the previous case, the changes would be applied only if the user clicked "OK", not if they clicked "Cancel".

If...Thens should be fairly easy to understand. IF the user clicks OK, THEN the code between "Then" and "End If" will run. There is an additional (optional) word that can be used, Else. For example (please don't put this code in your program),

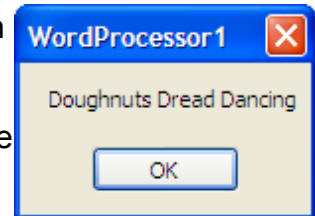
```
If FontDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
    RichTextBox1.SelectionFont = FontDialog1.Font
Else
    MsgBox("Doughnuts Dread Dancing")
End If
```

IF the user clicks OK, the code between the "Then" and "Else" will run. If the user does not OK (or instead clicks Cancel or the X to close the dialog), the code between "Else" and "End If" will run.



## The Message Box

A widely used function in Visual Basic is the message box, which simply brings a small dialog window up onto the screen, containing the text you specify. In the above example, if the user were to click Cancel instead of OK, a small box that looks like this would appear to inform the user that doughnuts dread dancing.



In addition to the text that the message box will display, you can also choose buttons to appear or the text to appear in the title. To specify which buttons to show you can add (for example) ', MsgBoxStyle.YesNo' after the last quote in that line of code but before the closing parenthesis. To specify a title, you can (for example) add ', "FYI:"' after ', MsgBoxStyle.YesNo' or ', , "FYI:"' if you have not specified buttons. In using a function, when there are a number of options that can be specified, each option has its own slot. If you do not wish to specify the type of MsgBoxStyle (which defaults to MsgBoxStyle.OK), you can type a comma and a space before moving on to the next option.

If you look at the Font code above, you may notice a problem. If the user has not selected any text and instead intends to change the font for the whole document, they will quickly become confused and frustrated. We need to write code to change the whole document's font. We can decide whether or not they intend to change the font of the selection or the whole document based on whether or not text is selected using the following code.

```
Dim FontDialog1 As New FontDialog
If RichTextBox1.SelectedText <> "" Then
    FontDialog1.Font = RichTextBox1.SelectionFont
    If FontDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        RichTextBox1.SelectionFont = FontDialog1.Font
    End If
Else
    FontDialog1.Font = RichTextBox1.Font
    If FontDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        RichTextBox1.Font = FontDialog1.Font
    End If
End If
```

You can see that we placed the code dimming of FontDialog1 before the If...Then statement because we want this line of code to run regardless of the selection. The <> in the second line means "not equal", so the code between the If and Else will be executed (run) if any text is selected. If no text is selected, or RichTextBox1.SelectedText is "" (empty), the code between the Else and End If will run.

## Font Color

From either the code view or the design view, prepare to enter the code that runs when the user clicks on View > Font Color. This code will be very similar to the Font code we entered at the beginning of this chapter.

```
Dim ColorDialog1 As New ColorDialog
If RichTextBox1.SelectedText <> "" Then
    ColorDialog1.Color = RichTextBox1.SelectionColor
```

```
    If ColorDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        RichTextBox1.SelectionColor = ColorDialog1.Color
    End If
Else
    ColorDialog1.Color = RichTextBox1.ForeColor
    If ColorDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        RichTextBox1.ForeColor = ColorDialog1.Color
    End If
End If
```

## Page Color

The page color can be changed, but it will not be saved in any format the RichTextBox can save in. Instead, the background color is something that would normally be saved in program preferences. However, it is easy to code.

```
Dim ColorDialog1 As New ColorDialog
If ColorDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
    RichTextBox1.BackColor = ColorDialog1.Color
End If
```

## Chapter 6: Input and Output using a Rich Text Box

- *Opening a File*
- *Closing a File*
- *Saving a File*
- *The Difference Between Save and Save As...*

### Opening a File

This can be achieved by choosing “Open” from the “File” menu of the Visual Studio IDE. Okay, that’s not what this section is about, but it does seem like a good time to mention that Microsoft Visual Studio .NET 2005 will allow you to open and often edit a large quantity of different file types.

From the design view, double-click File > Open (or select “OpenToolStripMenuItem” from the drop-down in the code view and select “Click” from the second drop-down). Enter this code (though you may read ahead before doing so):

```

If RichTextBox1.Text <> "" Then
    Call OpeningCode()
Else
    If MsgBox("Opening a new file will erase any text currently loaded.
Are you sure you want to continue?", MsgBoxStyle.YesNo, "Caution") =
MsgBoxResult.Yes Then
        RichTextBox1.Text = ""
        Call OpeningCode()
    End If
End If

```

At the second line of code, Visual Studio should tell you that OpeningCode() has not been defined. That is because OpeningCode() is a new Sub (procedure) I created separately. Place this code at the very end of all the code in a new line before “End Class”.

```

Private Sub OpeningCode()
    Dim OpenFileDialog1 As New OpenFileDialog
    OpenFileDialog1.Filter = "Plain Text Files|*.txt"
    If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        RichTextBox1.LoadFile(OpenFileDialog1.FileName,
RichTextBoxStreamType.PlainText)
    End If
End Sub

```

The reason I created a Sub instead of typing the code directly is because the same code has to be run twice. While I could duplicate the code, this makes it easier to fix any problems that might emerge. If I use a Sub instead of duplicating the code, I only have to fix a **bug** in a single place instead of many.

So the code at the beginning of the chapter reads: IF there is any text in RichTextBox1 (If RichTextBox1 is not empty), THEN the code inside of the sub OpeningCode() is run. Otherwise, the user is asked to verify that they want to erase its contents before loading the new file.

In OpeningCode() we dimension an OpenFileDialog (similar in function to the font and color dialogs) and set the filter for it. The filter determines which file types can be opened by the program, so the user does not try opening a picture as text (as doing so would cause lots

of errors). In the filter line of code, we give a description of files that can be opened – in this case “Plain Text Files”. The **syntax** requires a pipe (|) to separate the description of the extension from the actual extension. We then specify the extension by preceding the extension with an asterisk and a period, in this case “\*.txt” files. Please note that in Visual Basic (as many **computer languages**), the asterisk is a wildcard, meaning anything can fill that space. In this case, any file ending in “.txt” will be visible. However, if we wanted to tweak the filter to open Plain Text Files that start with the letter A, we could change “\*.txt” to “|a\*.txt”. We could also set the filter to open Plain Text Files that end with the letter A by changing “|\*.txt” to “|\*a.txt”.

## Closing a File

Closing a file is accomplished by asking the user if they would like to save any existing text before emptying the RichTextBox. Try writing some of this code on your own before referencing it below.

```

If RichTextBox1.Text <> "" Then
    If MsgBox("Would you like to save your progress before closing this
file?", MsgBoxStyle.YesNo, "Save?") = MsgBoxResult.Yes Then
        SaveToolStripMenuItem.PerformClick()
    End If
End If
RichTextBox1.Text = ""

```

The `SaveToolStripMenuItem.PerformClick()` instructs Visual Studio to pretend the user has clicked File > Save. It is an alternative to creating a Sub when you are referencing the same code in multiple places, because the code to save the file would be the same here as it would be if the user were to click File > Save.

## Saving a File

Saving a File is very similar to opening a file. Enter the code for the “Save As...” menu item (not “Save”).

```

Dim SaveFileDialog1 As New SaveFileDialog
SaveFileDialog1.Filter = "Plain Text Files|*.txt"
If SaveFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
    RichTextBox1.SaveFile(SaveFileDialog1.FileName,
RichTextBoxStreamType.PlainText)
End If

```

## The Difference Between Save and Save As...

Many programs have both Save and Save As options. The Save menu item saves the file without prompting for a location and will overwrite the existing file, whereas the Save As menu item asks where to save it. The following code excludes the save file dialog and replaces it with a variable that can be filled when a file is opened.

```

If CurrentFileLocation <> "" Then
    RichTextBox1.SaveFile(CurrentFileLocation,
RichTextBoxStreamType.PlainText)
End If

```

In order for this to work, we have to first define the variable `CurrentFileLocation`. Define it at the top of the code page, underneath "Public Class Form1" - defining it there will make the contents available to any Sub on the form. Define the variable as a string. We also have to adapt our opening code to populate the variable with the file location. This code is `CurrentFileLocation = OpenFileDialog1.FileName` and should be placed within the If statement in the `OpeningCode()` Sub. While we are at it, we should place `CurrentFileLocation = SaveFileDialog1.FileName` in the If statement of the `SaveAsToolStripMenuItem`. If the user decides to Save As... in order to save the file with a new name or in a different location, we need to make sure the Save menu item keeps track of this.

## Chapter 7: Manipulating Text

- *Using the Clipboard – Cutting, Copying, & Pasting*
- *Select All*
- *Finding Text*
- *Printing*

### Using the Clipboard – Cutting, Copying, & Pasting

The **clipboard**, as you may know, is a virtual layaway. It will store things like text to be used at a later point in the same or a different document. Cutting will store the text in the clipboard before deleting it from the document, copying text will simply store the text in the clipboard without deleting it from the document, and pasting text will place text from the clipboard into the document.

There are several ways of storing text on the clipboard. One way is to use `Clipboard.SetData` and specify the data type as text.

```
Clipboard.SetData(Windows.Forms.DataFormats.Text, RichTextBox1.SelectedText)
```

You can also use the `RichTextBox.Copy()` function, but the `SetData` function is useful to know because with it you can copy HTML, images, sounds, or other data as well as text. Using the `RichTextBox.Copy()` function, the code would look like this. You can choose which you would like to use.

```
RichTextBox1.Copy()
```

The code for cutting text is going to be almost identical to the code for copying. The only change is that when text has been selected, the selected text needs to be deleted from the document. To do this, add the line `RichTextBox1.SelectedText=""`. Alternately, you may use `RichTextBox1.Cut()`.

The easiest option for pasting is simply `RichTextBox1.Paste()`, though setting `RichTextBox1.SelectedText` equal to `Clipboard.GetData(Windows.Forms.DataFormats.Text)` is also an option.

### Select All

To select all of the text in a text box, simply use code similar to the following.

```
RichTextBox1.SelectAll()
```

### Finding Text

Before we get into the code for finding text, we need to declare two **global** variables `StringToFind` and `FindLoc` (so any Sub in the program can access them). To do this, declare them at the top of the code page, below `Class Form1` and below the other variable we declared globally, `CurrentFileLocation`. `StringToFind` needs to be dimmed as a string, and `FindLoc` needs to be declared as an integer.

```
FindLoc = 0  
StringToFind = InputBox("Enter the string to find", "Find", "")
```

```
FindNextToolStripMenuItem.PerformClick()
```

To find text, we need to ask the user for the string they wish to find. An `InputBox` is much like a `MessageBox` in that it is a small dialog that appears onscreen, the `InputBox` with a label (prompt) and a `TextBox`. The second line of code above sets `StringToFind` equal to whatever the user enters in an `InputBox`. The options that are specified in the `InputBox` are the prompt (“Enter the string to find”), the Title (“Find”), and default value. The default value is empty (“”) because while it is not a useful function of the `InputBox` now, it is important to know that you can specify the initial value. The third line of code instructs the program to run the 'Find Next' code.

The following code should run when the user clicks 'Find Next'.

```
Try
    FindLoc = RichTextBox1.Text.IndexOf(StringToFind, FindLoc)
    RichTextBox1.Find(StringToFind, FindLoc, RichTextBoxFinds.None)
    FindLoc += 1
Catch
    FindLoc = 0
End Try
```

The `FindLoc` variable will store the location of the last text found. `RichTextBox1.Text.IndexOf` will find the **index** of character in the text box, where the search string is found (“x” has an index of 3 in “text”). Because we have used `RichTextBox1.Text.IndexOf()` to find the desired text, the only purpose of `RichTextBox1.Find` is to highlight the targeted text. In the `RichTextBox.Find()` function, `StringToFind` is the string we are looking for, `FindLoc` is the index at which the search will begin, and `RichTextBoxFinds.None` specifies that the search is NOT case sensitive. `RichTextBoxFinds.MatchCase` can be used if you do want the search to be case sensitive. We set `FindLoc += 1` because we want the program to find the next instance of the search string if the user clicks 'Find Next' again.

The `Try...Catch` can be very useful, but also very risky. If anything goes wrong when the code between `Try` and `Catch` is run, the program will not give an error message, but instead run the code that appears between `Catch` and `End Try`. In this case, the `Try...Catch` is intended to prevent a specific error that will occur if the user continues searching for a string after the last instance has been found. In that case, the program needs to start searching back at the beginning to avoid an error. However, you need to be careful in using `Try...Catch`, because it is easy to add the `Try`, `Catch`, and `End Try` early on in programming and as a result miss an important bug when testing.

## Printing

I have chosen not to cover printing, because it is a fairly involved subject that does not need to be discussed in a gaming class. If you are feeling ambitious and would like to know how to print, feel free to ask, but for now, return to the design view and double-click the Form (the title bar, not the menu bar is a good place to do this). Any code you enter here will run when the Form first opens. Type `PrintToolStripMenuItem.Enabled = False`. This will make the Print menu item appear ghosted or gray, which indicates to the user that it is not available.

## Chapter 8: Leftover Menu Items

- *Undo & Redo*
- *Zooming*
- *Running External Programs (Help)*
- *Adding a Second Form (About this Program)*

### Undo & Redo

Undo and Redo are two functions that are built into the RichTextBox, but a similar effect can be created in games (such as card games) by creating and using stack variables. To use the built-in functions of the RichTextBox, simply use `RichTextBox1.Undo()` or `RichTextBox1.Redo()`. In fact, the hardest part of Undo and Redo are telling the Program when the options are available. In order to do this, we need to check whether or not each function is available (whether the user has made changes or not) when the user places their mouse over 'Edit'. To automatically generate the code for this, select `EditToolStripMenuItem` from the drop-down menu at the top of the code view and select `Click` from the second drop-down. Here, enter the code that follows.

```
If RichTextBox1.CanUndo = True Then UndoToolStripMenuItem.Enabled = True _
Else UndoToolStripMenuItem.Enabled = False
```

You can see from this example that `If...Then...Else` statements do not have to be on their own lines, but they can be on a single line, though. At the same time, you will notice the underscore character (`_`), which in Visual Basic is used to essentially join one line of code with the next. If there is a very long line of code, the underscore character can be used to break it up so you do not have to scroll right and left to see the whole line.

The code to detect if Redoing is a possibility is identical to the code for undoing, with the word 'redo' replacing 'undo'. Both sets of code need to be in the `EditToolStripMenuItem.MouseEnter` Sub before continuing.

### Zooming

Before writing the code to zoom, let's add a few items to our menu. To the right of `Zoom` (in the `View` menu), so the new options appear when the user clicks or hovers over 'Zoom', enter the following items.

```
Zoom &Out (Ctrl+OemMinus / Ctrl+-)
Zoom &In (Ctrl+Oemplus / Ctrl++)
&Custom Zoom
```

While in Visual Basic 6.0 and earlier the font had to be changed to make text bigger, in VB.NET the RichTextBox has a `ZoomFactor` property. This property is a decimal number, where 1.0 corresponds with 100% (normal size), .5 corresponds with 50% (half size), and 3.0 corresponds with 300% (triple size). The `Zoom Out` menu item should decrease the `ZoomFactor` from whatever its current value is by a set number, such as .1 (HINT: Use the `-=` operator from Chapter 4). `Zoom In` should do the opposite, increasing the `ZoomFactor` a set number. `Custom Zoom` should convert and set whatever value the user enters in an input box from a percent to the `ZoomFactor` decimal. Try to accomplish each of these three tasks on your own before referring to the examples on the next page.



```

Private Sub ZoomOutToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ZoomOutToolStripMenuItem.Click
    RichTextBox1.ZoomFactor -= 0.1
End Sub

Private Sub ZoomInToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ZoomInToolStripMenuItem.Click
    RichTextBox1.ZoomFactor += 0.1
End Sub

Private Sub CustomZoomToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CustomZoomToolStripMenuItem.Click
    Dim CZvar As String = InputBox("Please enter a percent of normal
view.", "Custom Zoom", "100%")
    If CZvar.EndsWith("%") Then
        RichTextBox1.ZoomFactor = CZvar.Substring(1, CZvar.Length - 1) /
100
    Else
        RichTextBox1.ZoomFactor = CZvar / 100
    End If
End Sub

```

The Custom Zoom code above creates a variable so that we can check the number entered for a percent sign. If the variable ends with the percent sign, we use the substring (only part of the string stored in the variable) instead of the whole string. In order to do this, we used `CZvar.Substring(1, CZvar.Length - 1)`. The first number is the index of where the substring should start, and the second number is the length to take for the substring. In this case, the second number is just the length of the string stored in the variable minus 1. In the end, we had to divide the number the user entered by 100 to convert the number from percent to `ZoomFactor` decimal.

It's okay if you did not plan on the contingency that the user might enter a percent symbol at the end of the number. In fact, the code above is not even foolproof. If the user enters any letters or punctuation other than % at the end, the program will produce an error. At this point you do not have to plan for every possibility, but you should begin to think about them.

### Running External Programs (Help)

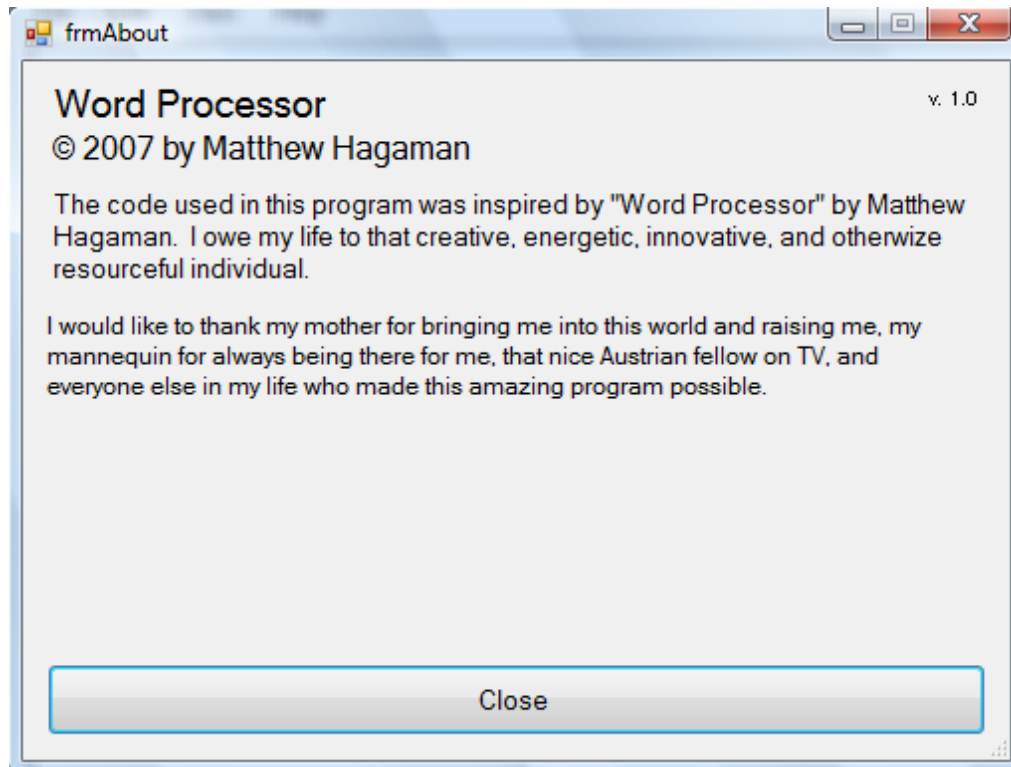
Usually the Help included in a program is actually a second program. If you were to make a second program to handle the Help files associated with your word processor you would use the code that follows. If you would like to, you should be able to easily alter the word processor you have already built to be a framework for handling help documents for any program you write in the future.

```
System.Diagnostics.Process.Start("StartHelpFile.txt", MyHelpProgram.exe)
```

The code above opens a specific file in a specific program. The same code without the `"StartHelpFile.txt"` would simply open the specified program. If you do not plan on adding a Help application, disable the Help menu item (not the heading, but the item) by changing the `Enabled` property to `False`.

## Adding a Second Form (About this Program)

To add a second form to your project, access the Solution Explorer and right-click your project's name. Choose Add > WindowsForm. In the dialog that appears, name your new form frmAbout.vb. Change the Size of your new form to "512, 384", set the Start Position to CenterScreen, and set MaximizeBox to False. Then enter some information about the program. This should include the name of the program, a version number, and a copyright notice. Any software you create can be protected by copyright. Also be sure to include a 'Close' button, which should initiate the code `Me.Close()` when clicked. Note that disabling the AutoSize option on labels will allow you to resize them and force the text to wrap.



The code to bring this window forth is fairly easy. When the user clicks on Help > About, enter this code to run.

```
frmAbout.ShowDialog()
```

## A Word About Version Numbers

Actually, more than one word. Version numbers can include numbers and letters. The initial retail version (the first final version) is typically version 1.0, with each earlier version a number smaller (commonly 0.2, 0.8, 0.95 based on progress towards the end product). As progress is made after the initial release, the version is often increased by 0.1 for every significant feature addition, .01 for every minor feature addition, and an incremented letter for each bug fix (i.e. 1.02b = initial version plus two small features and two bug fixes). Another alternative is basing the version number on a revision or beta version. Beta products are mostly-functional programs that may still contain a few bugs. Version numbers for these are typically something like 1.0beta1.

## Chapter 9: A Few Other Things

- *Other Events*
- *MDI: Multiple Document Interface*
- *Arrays*
- *0-based and 1-based indexes*
- *Modules*
- *Structures*
- *Do Loops*
- *For...Next Loops*
- *Mid*
- *Generating "Random" Numbers*
- *Working with Timers*
- *Focus*

### Other Events

If you have used the dropdowns at the top of the code page to automatically create the Subs for menu items, you should have noticed that there are a variety of other options besides click. I will highlight a few of them below.

**Activated** – When using multiple forms, and switching from one to another, the new or remaining form gets activated. Also, when switching between windows, the newly selected program will be activated. When either of these scenarios occurs, the code following the Activated event handler will run.

**CheckStateChanged** – When using CheckBoxes, CheckedListBoxes, or MenuItem's containing check boxes, anytime the value is changed (checked / unchecked) any code following the CheckStateChanged event handler will run.

**Closed** – When using multiple forms, if you hide one instead of closing it, you may need to place code in the closing command of the main form to end the program (or save the document, or...), otherwise the computer's memory may be monopolized by the program that should have been closed.

**DoubleClick** – If DoubleClick code is not entered, Click code should be run, but when you would like different things to occur when an object is clicked or double-clicked, you can enter code in the DoubleClick event handler.

**DragDrop** – You have probably dragged either text or images from one program to another or within a document. If EnableDragDrop is set to true and an object is dropped on the control, code following the DragDrop event handler will run.

**HScroll** – When the user scrolls left or right, code following the HScroll event handler will run.

**KeyDown** – While it may seem strange, Visual Basic interprets the hitting of a key (or mouse button) as three separate actions: Down, Press, and Up. If the KeyDown event is used with a specific key, the code would be run sooner than if KeyUp were used.

**Load** – Before the visual interface is loaded, code can be set to run. For example, if you wanted a message box to appear before your game is shown, you could enter code following the Load event handler.

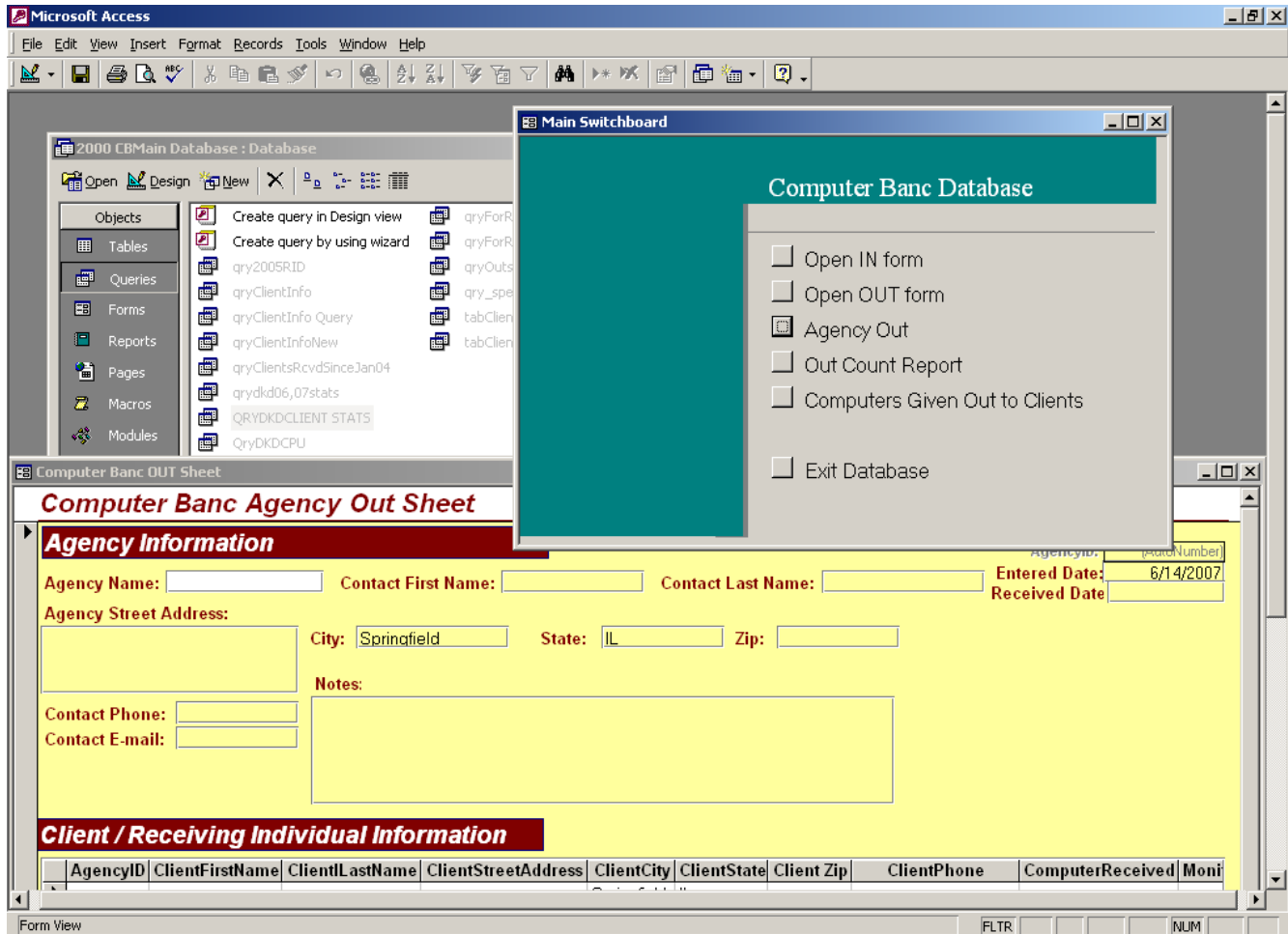
**MouseEnter** – MouseEnter, MouseHover, MouseMove, and MouseLeave all deal with the location of the mouse cursor. If you want the color of a button to change only when the mouse is over it, you could put code in both the MouseEnter and MouseLeave events.

**Paint** – When the form is loaded or when another window appears or is moved over the form, the visual interface has to be generated or regenerated. When this happens, any code following the Paint event handler is run.

**TextChanged** – When text is entered or changed in a TextBox or RichTextBox, code following the TextChanged event handler will run.

### MDI: Multiple Document Interface

MDI is an acronym that stands for Multiple Document Interface. As you might imagine, this is one way of having multiple documents open in the same program at the same time. Essentially, MDI gives you the ability to have multiple forms contained within a single form. The screenshot below is an example of a Microsoft Access database using MDI. MDI uses a system of hierarchy which consists of Parents and Children.



### Arrays

<a href="#">QuizQuestions(0)</a>
<a href="#">QuizQuestions(1)</a>
<a href="#">QuizQuestions(2)</a>
<a href="#">QuizQuestions(3)</a>
<a href="#">QuizQuestions(4)</a>
<a href="#">QuizQuestions(5)</a>
<a href="#">QuizQuestions(6)</a>
<a href="#">QuizQuestions(7)</a>
<a href="#">QuizQuestions(8)</a>
<a href="#">QuizQuestions(9)</a>

Up to this point, we have been working with variables that can only hold a single value. Arrays are a neat way of storing multiple values in the same variable. Think of arrays as tables. An array declared `QuizQuestions(9)`, left, would be a single variable that can hold 10 values. *It will hold ten values instead of nine because arrays have 0-based indexes* (see the next section).

Arrays can be multi-dimensional, meaning they can be two-dimensional (contain columns and rows) or have

<a href="#">QuizQuestions(0, 0)</a>	<a href="#">QuizQuestions(1, 0)</a>	<a href="#">QuizQuestions(2, 0)</a>
<a href="#">QuizQuestions(0, 1)</a>	<a href="#">QuizQuestions(1, 1)</a>	<a href="#">QuizQuestions(2, 1)</a>
<a href="#">QuizQuestions(0, 2)</a>	<a href="#">QuizQuestions(1, 2)</a>	<a href="#">QuizQuestions(2, 2)</a>
<a href="#">QuizQuestions(0, 3)</a>	<a href="#">QuizQuestions(1, 3)</a>	<a href="#">QuizQuestions(2, 3)</a>
<a href="#">QuizQuestions(0, 4)</a>	<a href="#">QuizQuestions(1, 4)</a>	<a href="#">QuizQuestions(2, 4)</a>
<a href="#">QuizQuestions(0, 5)</a>	<a href="#">QuizQuestions(1, 5)</a>	<a href="#">QuizQuestions(2, 5)</a>

even more dimensions.

The array declared `QuizQuestions(2, 5)`, above, is a single variable that can hold 18 values in 3 columns ( $2+1=3$ ) and 6 rows ( $5+1=6$ ).

### 0-based and 1-based Indexes

Indexes, which we have discussed in connection with the location of characters in a string (i.e. "x" has an index of 3 in "text") and in the creation of arrays (i.e. the index of `NewArray(3)` is 3), can be structured in two ways. While many computer languages use one system of indexing or the other, Visual Basic uses both: 0-based and 1-based. The difference is basically the way the language interprets counting: starting and including 0 (as a computer would count) or starting and including 1 (as a human would count). *In most instances, it is true that text elements are 1-based and other elements are 0-based, but the ToolTip that appears when typing code for a specific function should tell you the system to use* (see ToolTip on page 11 for a description of a ToolTip).

### Modules

When using multiple forms, you may be required to transfer the values of variables between forms. In order to accomplish this, you may create a Module in which you can type standard VB code which will be available to all forms. Modules can be used for both public variables and public functions (Subs), but *both have to be defined as Public*.

### Structures

Similar to arrays, structures are useful in storing multiple variables. However, while arrays must use an index, structures contain named variables. The following is an example of code that makes up a structure, in this case the structure named 'Pepsi' has a variable for each ingredient.

```
Structure Pepsi
    Dim HighFructoseCornSyrup As Decimal
    Dim Sugar As Decimal
    Dim Colorings As Decimal
    Dim PhosphoricAcid As Decimal
    Dim Caffeine As Decimal
    Dim CitricAcid As Decimal
    Dim NaturalFlavors As Decimal
    Dim Water As Decimal
End Structure
```

Another advantage to structures is that they are designed as templates. You can declare an infinite number of variables that contain the variables defined within the structure. This structure must be defined outside of any Sub that references it.

The following code to utilize the structure `Pepsi` to create 4 variables using that structure as a template, which could each contain independent values. The second line of code that follows is the code that would be used to recall the value stored in one of the structure's variables.

```
Dim Pepsi(3) As Pepsi

TextBox1.Text = "Caffeine: " & Pepsi(0).Caffeine
```

## Do Loops

Loops are a very useful element of any computer language. They can make the programmer's job much easier, by performing the same code over and over.

```
Do While Condition
    Statement(s)
Loop
```

```
Do
    Statement(s)
Loop While Condition
```

```
Do Until Condition
    Statement(s)
Loop
```

```
Do
    Statement(s)
Loop Until Condition
```

There are two different types of Do loops, one which runs as long as a condition is met and another which runs until a condition is met. These syntaxes are commonly used when there is no way to predict how many times code needs to be run for the condition to occur. For example, if characters are doing battle and the amount of damage done is computed with a random number (see Generating “Random” Numbers in this chapter), you might have code that would...

```
Do
    Attack
Loop While EnemyAlive = True
```

However, it is more common to use For...Next Loops in situation when you know or can calculate how many times code needs to be run.

## For...Next Loops

For...Next Loops are generally used more frequently than Do Loops. The For...Next Loop is set up as follows.

```
Dim i As Integer
For i = 0 To ListBox1.Items.Count - 1
    RichTextBox1.AppendText(ListBox1.Items.Item(i) & ChrW(10))
Next
```

The For loop always includes a counter, which can be a pre-existing variable or a variable created only to be used in the counter. In this case, *i* was dimensioned for the purpose of counting. In this case, the code inside the For...Next Statement will run once for each item inside the ListBox. (The counting starts at 0 and ends at Count – 1 because the item indexes are 0-based).

- *The RichTextBox.AppendText function will add anything within its parentheses to the very end of whatever is already in the text box.*
- *ChrW(10) refers to the 'Return' or 'Enter' character, bumping any text following it to the next line.*

## Mid

Similar to substring (and available in Visual Basic before Substring) is the `Mid` function, which allows us to access only part of any string. The syntax is

```
Mid(TextPropertyOrVariableName, Start Position, Length)
```

The `TextPropertyOrVariableName` can be something like `RichTextBox1.Text` or `StringVariable1`. The start position is 1-based. The length is optional, so if you only include the string and start position, the `Mid` function will capture each character from the start position through the end.

## Generating “Random” Numbers

Because computers cannot imagine (but can only run calculations), any random number the computer comes up with is really not random. In fact, they are often referred to as pseudo-random. But for most purposes, the random numbers Visual Basic generates are random enough. The `Random` function, `Rnd()`, will generate numbers anywhere from 0.000...001 to 0.999...999 (or between 0 and 1). In order to change these numbers into numbers you can use, you may have to multiply the number produced by 10, 100, or 1000 and then use the `CInt()` function to convert the number from a decimal to an integer (whole number).

- The `CInt()` function simply cuts the end off of any decimal number in the parentheses to convert it into an integer.
- There are other 'C' functions such as `CDate()` with similar functionality.

## Working with Timers

There are just a few things to know about timers. Timers are disabled when added. In order to make them functional, they must be enabled. The code for this is `Timer1.Enabled = True`. Make sure to disable the timer when it has performed its role, otherwise it will result in an endless loop. The duration (or interval) is recorded in milliseconds, so an interval of 1000 would last for one second.

## Focus

It is easy to help the user navigate your form using the `Focus()` function. It tells the computer to give the specified control priority. For example, if a `TextBox` is targeted the cursor will appear in that `TextBox`. If a `Button` is targeted, the user can hit the Enter key to quickly click it.

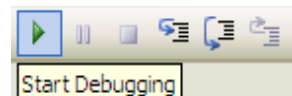
## Chapter 10: Debugging

- *Breakpoints*
- *Viewing the Values of Variables*

**Debugging** is the process of **compiling** and running your program with the intention of checking it for errors or bugs. Visual Studio has a number of built-in features that can help you in the debugging process.

### Breakpoints

The most important of these is breakpoints, which force the program to run only until it reaches a certain point in the code. To set or remove a breakpoint, click the gray bar just to the left of the line of code at which you would like to break (pause). When run, the program will pause *before* the selected line. To start debugging, press F5 or click the Start Debugging button on the Visual Studio toolbar. The program will run normally until it reaches the line of code at which you set at a breakpoint. If this line was under an event handler requiring user action, you will need to perform that action before the program will stop. The yellow highlight indicates the line of code that will be run next. To continue the program step by step you may use the 'Step Into' button (the first with the blue arrow in the screenshot above), or if you would like to continue normally you may press the Start Debugging button, which has now become the Continue button.



### Viewing the Values of Variables

Anytime during debugging, you can minimize your program to get back to the Visual Studio interface. There, you can see the value of any variable by simply hovering the mouse over the variable. Just note that if there is a variable with the same name in more than one Sub, you can hover over the variable in a sub where the code is not running and it will show you a value, but this value refers to the variable in the currently active Sub.



## Chapter 11: Working With GIMP

- *Where to Get GIMP*
- *At First Glance*
- *Drawing Tools*
- *Using Filters*
- *Saving and Exporting Pictures*

### Where to Get GIMP

GIMP is a free image manipulation program (GIMP stands for GNU(free) Image Manipulation Program) which has become a popular replacement for Adobe Photoshop. The newest version of GIMP can be downloaded from [www.gimp.org](http://www.gimp.org). While gimp.org does not host its own compiled program (only the source code), there is a link from the main page labeled GIMP for Windows which will allow you to download all of the components necessary. The first of these is GTK+, which is a framework of sorts on which the visual components of GIMP are built. The primary package is GIMP itself, but there is also a documentation package (with help files) and an animation package, which is optional but could be used to add an element of movement to a game.

### At First Glance

At first glance, GIMP may seem confusing. For starters, there several windows. The first is the main window, from which you can open and create new files. This main window also contains most of the tools we will be working with, including buttons for the brush, pencil, eraser, and paint can tools. The second window gives you access to layers and many options for various tools. The window that appears when you create a new file has many menu items that will allow you to save, change size and resolutions, and play with filters.

### Drawing Tools

To get started, create a new file following the first template option (640x480). Experiment with the pencil and paintbrush tools (the difference being the greater precision of the pencil), being sure to check out the brush options at the bottom of the main window. I find the Sparks and Pencil Sketch brushes to be particularly interesting.

Brushes can be used in a number of ways: to draw on top of a background, to draw “dissolved” on top of a background, etcetera.

Opacity will allow the brush to draw on top of existing artwork and allow some percentage of that existing artwork to show through. Fade-out effects allow the brush pattern to fade out over a specified distance.

There are two color options in the main window, which appear as one square on top of another. The top is the fore color, or the color that will be used for the brush or text you are currently working with. The bottom square is the background color, which may be applied to the whole background from the Edit menu or may come into play with some effects.

### Using Filters

An easy way to make your artwork look more professional is by applying a filter. While some filters will change your artwork entirely, others will make subtle changes.

## **Saving and Exporting Images**

Because Visual Studio is a program made by Microsoft Corporation, it prefers native file formats over other proprietary formats. What this means is there are two ways to save your artwork to be imported into Visual Studio. The highest quality is a Windows **BMP image**, which has a file extension of \*.bmp. The format that should be used when distributing your game is a **JPEG image**, which has the extensions of \*.jpeg, \*.jpg, and \*.jpe. I would advise that you save your artwork in two formats, always in the native GIMP format of \*.xcf and also as either a BMP or JPEG image. The first is the best format to save in for later editing, and the other two are easy to import into Visual Studio.

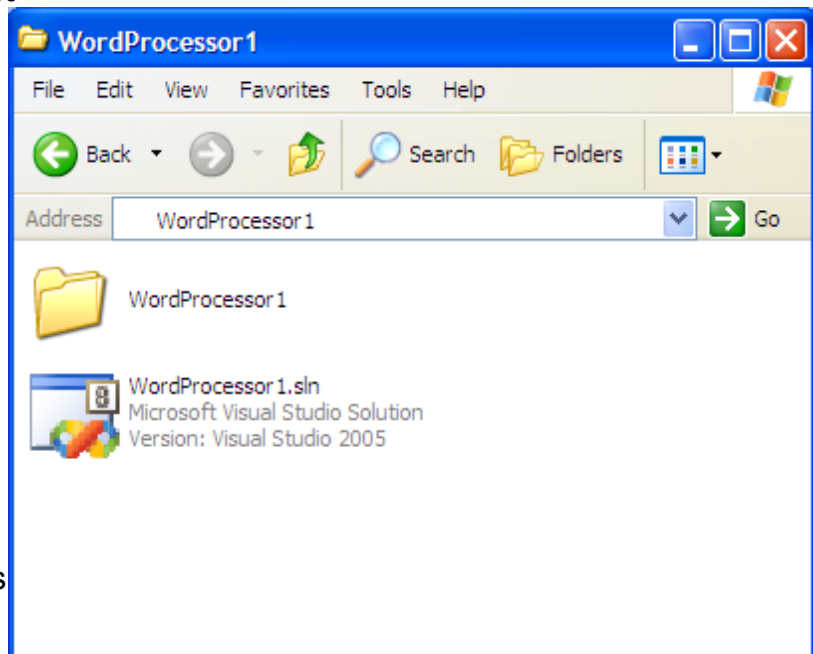
## Chapter 12: Distribution

- *Moving Your Visual Studio Project*
- *Forms of Distribution*
- *Install Creator*
- *Patch Maker*

### Moving Your Visual Studio Project

First thing's first. Moving the project itself from one computer to another does not involve just moving a single file. The project itself consists of several folders within folders with files and more files. To transfer the project itself, you must copy the folder that contains files like the image on the right. There should be a file with the extension of \*.sln (Visual Studio Solution) and a folder, both named after the project.

In addition, you may have to copy files such as images that are not located in the project folder. It is recommended that you create a folder within the project folder to hold things like images you wish to include in your program so moving the project is easier and everything is kept together for future reference.



### Forms of Distribution

Currently, the two most popular forms of distribution are via disc and via the Internet. In both of these methods, it is easy use the same installation file. In order to install your program, you generally want to create an installer. This program should put everything into one file to distribute which will in turn copy the required files where they need to go.

### Install Creator

Install Creator is a great program available from [www.clickteam.com](http://www.clickteam.com) which will create an installer for you. The program has a free version which is virtually the same as the for-cost version, the only difference is an advertisement for Install Creator will display at the conclusion of your installation.

### Patch Maker

If you are like every other programmer, you may find that you have to change your program to reflect a new bug or a must-have feature. In order to do so, you might create a whole new installer. Patch Maker, also available from [www.clickteam.com](http://www.clickteam.com), will create an installer that will install only the files that have changed from the original installation, resulting in a smaller and more efficient installation.

## Appendix 1: Glossary

**.NET Framework:** The package that must be installed before any program created with Visual Basic .NET can run.

**Access Databases:** The database format supported by Microsoft Access, it is a relatively simple format for developers to use if they need to access large quantities of organized data. Access Databases typically have the file extension of \*.mdb.

**Application:** The software program that the end user sees. Applications typically have the file extension of \*.exe. See Software.

**ASP .NET (Active Server Pages):** An architecture that allows the developer to create series of web pages that are programmed like Windows applications but appear to the user as HTML (that is, normal) web pages.

**BMP (Bitmap) Image:** This image file format saves the exact color (in RGB, amount Red, Green, and Blue, format) of every pixel in the image. Bitmap images typically have the file extension of \*.bmp. See pixel.

**Boolean:** A data type that can contain one of two values: true or false.

**Bug:** An error in program code that results in an unintended (or the lack of an) action. When the user enters a word into a field that should only hold numbers and the program does not have a predefined response, a bug has emerged. See field.

**Char:** A char variable will store a single character, be it letter, number, or punctuation mark.

**Clipboard:** The clipboard is a place in the computer's memory where you can temporarily store files, pictures, text, etcetera by using the Cut, Copy, and Paste functions in many programs.

**Computer Language:** A language (such as Visual Basic) that allows a programmer to communicate with a computer much like English allows one person to talk to another. A computer language, like any spoken language, has a set syntax and vocabulary. See syntax.

**Compiling:** In layman's terms, compiling is the process the computer uses to convert your layout design and code into a single application. Visual Basic is not the only language that compiles: every computer language has to be converted from human-understood code to 1s and 0s that a computer can understand.

**Control:** A general name for any item the user can interact with that is placed on a form. Examples include buttons, menus, and text boxes.

**Debugging:** The name given to the process of running your program and letting Visual Studio identify errors that occur during normal use of your program.

**Dialog:** A window or other visual element that appears on the screen on top of the main program, allows the user to enter or read information, then closes to return focus to the main program. See focus.

**Disposed:** When a control or variable is no longer needed, it can be permanently removed from the program and dismissed from memory. This is also known as disposal.

**Distribution:** 1. Another way to refer to an installation package, a distribution includes all the files that must be present on a user's computer for a given program to run.  
2. The process of transferring a program to the end user.

**Field:** 1. A control where data can be entered (such as a text box)  
2. A component of database hierarchy.

- Focus:** When you switch from one program to another, such as from a word processor (i.e. Microsoft Word) to a web browser (i.e. Microsoft Internet Explorer), you give the new program focus. The program with focus is the one that will capture anything the user enters on the keyboard.
- Form:** In most basic terms, form is another name for screen real estate, or the space that you have to place items on. Form also translates to 'window', as you may have multiple forms in a given application.
- Function:** A process built into Visual Studio in order to process data or otherwise accomplish a goal. Examples of functions include `RichTextBox.Find()`, `Text.Substring()`, `Rnd()`, and `ListBox.Items.Clear()`.
- Global Variable:** A term used to describe two types of variables.
1. A variable that appears outside of any Subs on a Form, but within the Class declaration.
  2. A variable declared publicly in a module, thus being accessible to all Forms.
- Hardware:** A term used to describe any piece of the computer that can be physically added or removed. Examples include add-in cards, hard drives, CD-ROM drives, etc.
- HTML (Hyper Text Markup Language):** The most widely used format for web pages.
- Icons:** Small pictures used to indicate a button or program's function (usually 32 pixels wide and 32 pixels tall).
- IDE (Integrated Development Environment):** A name given to the program used by developers to create applications; in this case Visual Studio.
- Index:** 1. A number that refers to the ranking of a character (or string of characters) within a body of text. For example, "x" has an index of 3 in "text".
2. A number that refers to the location of an element in an array.
- Integer:** A whole number data type.
- JET Databases:** See Access Databases.
- JPEG (Joint Photographic Experts Group):** This image file format stores the exact color of some of the pixels and then approximates the rest. In addition, this information undergoes compression to make the file size even smaller. JPEG files typically have the file extension of \*.jpg. See pixel.
- Pixel:** The smallest distinguishable dot on a computer monitor or printed media. There are 786,432 pixels on a standard-resolution (1024x768) monitor.
- Plain Text:** A text file format that supports only text; not multiple fonts or font styles, tables, or images. Plain Text files typically have the extension of \*.txt.
- Proprietary:** A proprietary format is a format owned and licensed by a single company. You can only write software to manipulate Microsoft Word Documents if you license it from Microsoft, but you can write software to manipulate Rich Text Format Documents without licensing it because it is NOT proprietary.
- RTF (Rich Text Format):** A text file format supporting bold and italic text, tables, and images. RTF documents may be opened by nearly all word processing programs and is often used when sharing documents. Rich Text files typically have the extension of \*.rtf.
- Software:** A term used to describe any application or program on your computer that cannot be physically separated from the computer, but is stored on your computer's hard drive. See hardware.
- Stack:** A variable that stores and recalls data in the order in which it is added. An excellent

example of use for this type of variable is in a web browser. When you click any link, the destination is added to the top of the 'Back' stack. When you click 'Back', you take the last destination from the top of the stack, load it, and add it to the top of the 'Forward' stack. When you click 'Forward', you take the page at the top of the 'Forward' stack, load it, and add the page's **URL** to the top of the 'Back' stack.

**String:** A text data type. If only one character is required, the char data type may be used.

**SQL Server:** Pronounced Sequel Server, this is the proprietary framework Microsoft has produced for creating advanced network connections for the management of data and databases.

**Syntax:** The grammar of a language. Just as someone would not fully understand another person who never used nouns, the computer cannot understand the programmer if their communication is in an improper format. In VB, for example, if an equal sign is used, it must be followed by a property, variable, or function that produces a value.

**URL:** Universal Resource Location. The location of a page, image, or video on the Internet.

**Variable:** A variable is a named space in your computer's memory that stores data.

**VB (Visual Basic):** A computer language used to create Windows applications or programs. In this book, the version that is used is Visual Basic .NET 2005.

## Appendix 2: Recommended Software & References

### Recommended Software

**Blender:** A free CAD (Computer Aided Design) program useful in creating textured or 3-dimensional computer models. While using 3-dimensional graphics within Visual Basic is extremely difficult at best, you can use images of 3-dimensional graphics created in Blender to spice up your programs. [www.blender.org](http://www.blender.org)

**GIMP (GNU Image Manipulation Program):** A free photo editing program which has become a popular replacement for Adobe Photoshop. An important asset if you are creating an image-driven game. [www.gimp.org](http://www.gimp.org)

**Install Creator:** Creating an installer is the easiest way to distribute your applications. You can use Install Creator and use it for free to distribute programs quickly and easily as a single-file installer. <http://www.clickteam.com>

**OpenOffice.org:** OpenOffice.org Writer is a free word processing program which has become a popular replacement for Microsoft Word. Its package also includes Calc, Impress, Base, and Draw, which are replacements for Microsoft Excel, Microsoft Power Point, Microsoft Access, and Microsoft Publisher, respectively. OpenOffice.org will open and save to a variety of file formats, including Microsoft's. [www.openoffice.org](http://www.openoffice.org)

**Patch Maker:** Patch Maker is a program designed to supplement Install Creator that will compare files from the original installation to files from an updated installation and create an installer that will change only the required files. This will produce a smaller installation file than a completely new installer would. <http://www.clickteam.com>

**Scribus:** A free desktop publishing program that is emerging as a replacement for Microsoft Publisher. It can be used to plan designs of web pages or print media. [www.scribus.net](http://www.scribus.net)

### References

**Introducing Microsoft Visual Basic 2005 for Developers** by Microsoft. Available from <http://msdn2.microsoft.com/en-us/vbasic/ms789094.aspx>.

**Microsoft Visual Basic .NET 2005 Step by Step** by Michael Halvorson. Published in 2005 by Microsoft Press; Redmond, Washington 98052-6399.

**Microsoft Visual Basic .NET Step by Step** by Michael Halvorson. Published in 2002 by Microsoft Press; Redmond, Washington 98052-6399.