Welcome to Visual Basic .NET

The goal of this book is to help you come up to speed with the Visual Basic .NET language even if you have never programmed anything before. We will start slowly, and build on what we learn. So take a deep breath, let it out slowly, and tell yourself you can do this. No sweat! No kidding!

Programming a computer is a lot like teaching a child to tie their shoes. Until you find the correct way of giving the instructions, not much gets accomplished. Visual Basic .NET is a language in which you can tell your computer how to do things. But like a child, the computer will only understand if you explain things very clearly. If you have never programmed before, this sounds like an arduous task, and sometimes it is. However, Visual Basic .NET gives you a simple language to explain some very complex things. Although it never hurts to have an understanding of what is happening at the lowest levels, Visual Basic .NET frees the programmer from having to deal with the mundane complexities of writing Windows programs. You are free to concentrate on solving problems.

Visual Basic .NET helps you create solutions that run on the Microsoft Windows operating system. Chances are, if you are looking at this book, you have already felt the need or the desire to create such programs. Even if you have never written a computer program before, as you progress through the Try It Outs in this book, you will become familiar with the various aspects of the Visual Basic .NET language, as well as its foundation in Microsoft's .NET Framework. You will find that it is not nearly as difficult as you have been imagining. Before you know it, you will be feeling quite comfortable creating a variety of different types of programs with Visual Basic .NET. As the name implies, Visual Basic .NET can be used to create applications for use over the Internet. However, as when learning any new technology, you have to walk before you can run, so we will start out focusing on Windows applications, before extending our boundaries.

Windows Versus DOS Programming

A Windows program is quite a bit different from its ancient relative, the MS-DOS program. A DOS program follows a relatively strict path from beginning to end. Although this does necessarily limit the functionality of the program, it also limits the road the user has to take to get to it. A DOS program is like walking down a hallway; to get to the end you have to walk down the hallway, passing any obstacles that you may encounter. A DOS program would only let you open certain doors along your stroll.

Windows on the other hand, opened up the world of **event-driven programming**. Events in this context include, for example, clicking on a button, resizing a window, or changing an entry in a textbox. The code that you write responds to these events. To go back to the hallway analogy: in a Windows program to get to the end of the hall, you just click on the end of the hall. The hallway can be ignored. If you get to the end and realize that is not where you wanted to be, you can just set off for the new destination without returning to your starting point. The program reacts to your movements and takes the necessary actions to complete your desired tasks. Visual Basic .NET simplifies the process of writing the code to handle each event by allowing the programmer to write code only for those events that mean something in the context of the program. All other events are ignored. For example, Windows distinguishes clicks from double-clicks, which means that if you only want your program to react to a single click, you need only write code for this single click; you do not have to write code to handle both single and double-click events.

You have probably already begun to suspect that the hallway analogy is grossly over-simplified. The main idea here is that in the DOS world, the user reacts to the program, and in the Windows world, the program reacts to the user.

Another big advantage in a Windows program is the **abstraction** of the hardware. What this means is that Windows takes care of communicating with the hardware for you. You do not need to know the inner workings of every laser printer on the market, just to create output. You do not need to study the schematics for graphics cards to write your game. Windows wraps up this functionality by providing generic routines that communicate with the drivers written by the manufacturers of hardware. This is probably the main reason why Windows has been so successful. The generic routines are referred to as the **Windows API** (**Application Programming Interface**).

A Brief History of Visual Basic

Before Visual Basic 1.0 was introduced to the world in 1991, developers had to be well versed in C++ programming, as well as the rudimentary building blocks (Windows API) of the Windows system itself. This complexity meant that only the dedicated and properly trained were capable of turning out software that could run on Windows. Visual Basic changed all of that, and it has been estimated that there are now as many lines of production code written in Visual Basic as in any other language.

Visual Basic changed the face of Windows programming by removing the complex burden of writing code for the **user interface** (**UI**). By allowing programmers to *draw* their own UI, it freed them to concentrate on the business problems they were trying to solve. Once the UI is drawn, the programmer can then add code to react to events.

Visual Basic has also been extensible from the very beginning. Third-party vendors quickly saw the market for reusable modules to aid developers. These modules, or controls, were original referred to as **VBXs** (named after their file extension). If you didn't like the way a button behaved you could either buy or create your own. However, these controls had to be written in C or C++. Database access utilities were some of the first controls available.

When Microsoft introduced Visual Basic 3.0, the programming world changed again. Now you could build database applications directly accessible to users (so called **front-end applications**) completely with Visual Basic. There was no need to rely on third-party controls. Microsoft accomplished this task with the introduction of the **Data Access Objects** (**DAO**), which allowed programmers to manipulate data with the same ease as manipulating the user interface.

Versions 4.0 and 5.0 extended the capabilities of version 3.0 in order to allow developers to target the new Windows 95 platform. Crucially they also made it easier for developers to write code, which could then be manipulated in order to be used by other language developers. Version 6.0 gave us a new way to access databases with the integration of **ActiveX Data Objects** (**ADO**). ADO was developed by Microsoft to aid web developers using Active Server Pages to access databases. With all of the improvements to Visual Basic over the years, it ensured its dominant place in the programming world. It helps developers write robust and maintainable applications in record time.

With the release of Visual Basic .NET, many of the restrictions that used to exist have been obliterated. In the past, Visual Basic has been criticized and maligned as a "toy" language, as it did not provide all of the features of more sophisticated languages such as C++ and Java. Now, Microsoft has removed these restrictions and made Visual Basic .NET a very powerful development tool. Visual Basic .NET has become a great choice for programmers of all levels.

Installing Visual Basic .NET

You may own Visual Basic .NET:

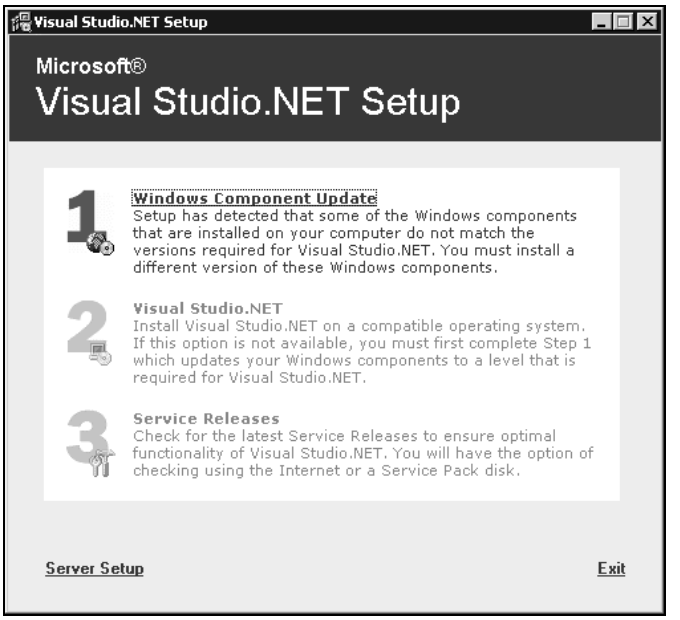
- As part of **Visual Studio .NET**, a suite of tools and languages that also includes **C#** (pronounced C-sharp) and **Visual C++ .NET**
- Standard Edition, which includes a cut down set of the tools and languages available with Visual Studio .NET

Both enable you to create your own applications for the Windows platform. The installation procedure is straightforward and easy to do. In fact, the Visual Basic .NET Install is smart enough to figure out exactly what your computer requires in order to make it work.

The descriptions that follow are based on installing Visual Studio .NET. However, all of Visual Studio .NET's languages use the same screens and windows (and hence look very similar), so you won't be seeing much that you would not see anyway.

Try It Out – Installing Visual Basic .NET

1. The Visual Basic .NET CD has an auto-run feature, but if the Setup screen does not appear after inserting the CD, then you will have to run `setup.exe` from the root directory of the CD. To do this, go to your Windows Start menu (usually found right at the bottom of your screen) and select Run. Then type d:\setup.exe into the Open box, where d is the drive letter of your CD drive. After the setup program initializes you will see the following screen:



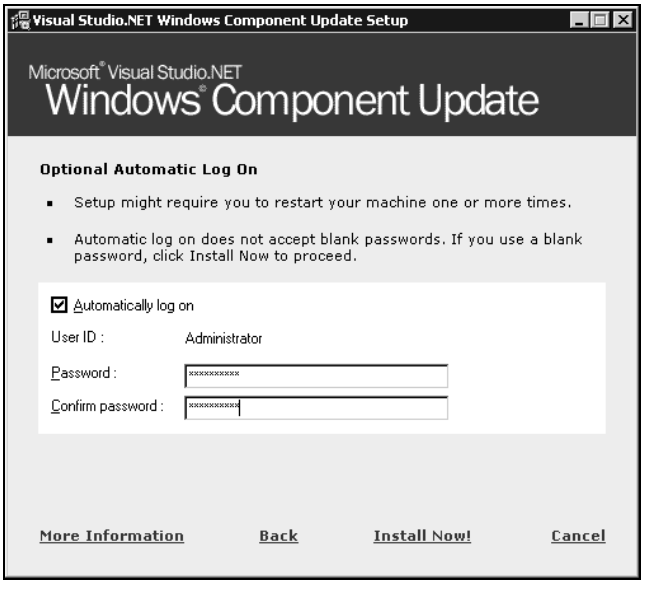
2. This dialog box shows the order in which the installation takes place. In order to function properly, Visual Basic .NET requires that several components and updates be installed on your machine. Step 1 is the Windows component update, so click on the Windows Component Update link; you will then be prompted to insert the Component Update CD that came with your Visual Studio .NET disks.

3. The installation program will then examine your system to see exactly which components have to be installed. Depending on the current state of your machine, this list could include any of the following items:

- Windows NT 4.0 Service Pack 6.0a
- Windows 2000 Service Pack 2
- Windows Installer 2.0
- Windows Management Infrastructure
- FrontPage 2000 Web Extensions
- FrontPage 2000 Server Extensions
- Setup Runtime Files
- Internet Explorer 6.0 and Internet Tools
- Microsoft Data Access Components 2.7
- .NET Framework

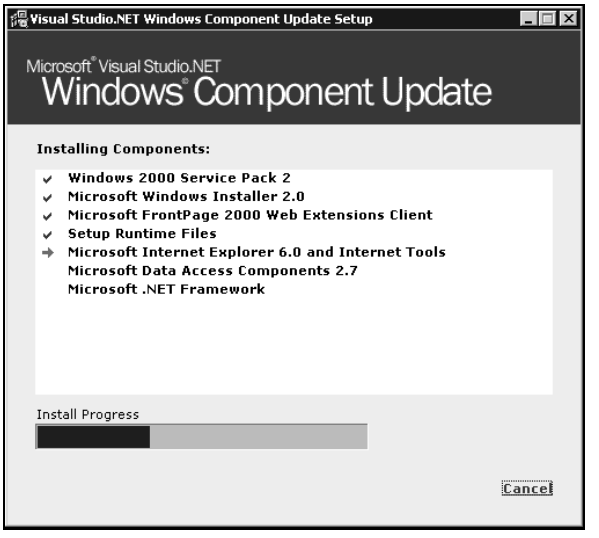
If you don't know what some of those things are, don't worry about it. They are just Windows components that Visual Studio .NET or Visual Basic .NET requires.

4. There may be numerous reboots of your computer as the system components are updated. However, Microsoft has added a nifty automatic login feature to minimize the time required to watch the installation if your computer requires a password to login:



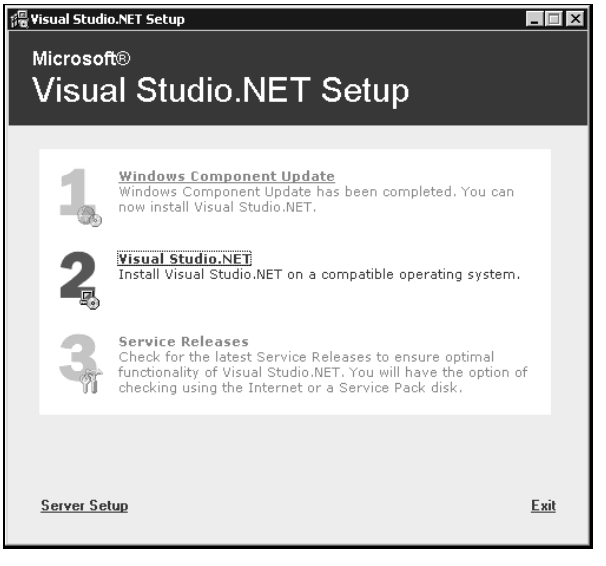
As you can see, this feature is optional. If you choose not to use it, you will be prompted to login after every reboot. This feature could provide a possible security breach, as the person who is actually logged in does not have to be sat at the computer (so you may want to avoid using this option in a busy office, for example), but it can reduce the boredom of watching installation progress screens.

5. Click on Install Now! and the component update will begin. Completed items will have a check mark and the component currently being installed will be identified by a red arrow:



Depending on what components you already have installed on your machine, your list of components that require updating may be different. For reference, these are the options on a fresh install of Windows 2000 Professional, with no other software.

6. After the Windows Component Update has finished, you will be returned to the Setup screen once more. You will now be able to install Visual Studio .NET, so click on Visual Studio.NET:



If at some point you wish to develop Web applications on this same machine, you will need to have Internet Information Services (IIS) and FrontPage Extensions installed. These are installed by default on Windows 2000 Server however, when installing on Windows 2000 Professional you may encounter the following screen:

Selecting Install Components *will guide you through the process of getting IIS and FrontPage 2000 Server Extensions installed. You will then be able to continue installing Visual Basic .NET.*

If you choose Continue, you will not be able to develop local Web applications (such as those discussed in Chapters 17 and 19) and will require access to a Web server.

7. As with most installations you will be presented with an option list of components to install. You can just install the features that you need. For example, if your drive space is limited and you have no immediate need for Visual C++ .NET, you can exclude it from the installation. You will also be given the chance to select the location of items (although the defaults should suffice unless your particular machine has special requirements). Any options not chosen at the initial setup can always be added later as your needs or interests change.

There are three sections of information given for each feature:

- The Feature properties section outlines where the required files will be installed and how much space will be needed to do this
- The Feature description box gives you an outline of what each feature is and does
- Finally, the Space Allocation section illustrates how the space on your hard drive will be affected by the installation as a whole

When you are running Visual Basic .NET, a lot of information is swapped from the disk to memory and back again. Therefore, it is important to have some free space on your disk. There is no exact rule for determining how much free space you will need, but if you use your machine for development as well as other tasks, anything less than 100MB free should be considered a full disk.

8. Visual Studio .NET includes a large set of documentation files, the settings for which are displayed under MSDN Documentation. The documentation can be set to run from the CD, or installed directly on to your hard disk. You can set the install location as Path (install to your hard disk) or Run from source (to have the documentation remain on the CD), via the Feature properties box. Each piece of the documentation can be configured in this manner.

   The default is for the documentation to remain on the CD. Only the indexes are copied to your hard drive to aid in searches. This means that you will have to have the CDs available in order to access the documentation. You are free to install some or all of MSDN onto your hard drive. The following table lists the approximate sizes of each of the document sets by their installed location:

	Hard Disk Space Required	
	Docs on CD	**Docs on Hard Drive**
Visual Basic Documentation	0.02 MB	5.93 MB
Visual C# Documentation	0.01 MB	1.48 MB
Visual C++ Documentation	0.04 MB	22.84 MB
.NET Framework Documentation	0.02 MB	39.70 MB
Platform SDK Documentation	0.31 MB	137.21 MB
Additional MSDN Documentation	0.17 MB	467.99 MB
Visual Studio Tools Documentation	0.05 MB	11.22 MB
Visual Studio Documentation	0.03 MB	9.90 MB
Knowledge Base Articles	0.14 MB	114.30 MB

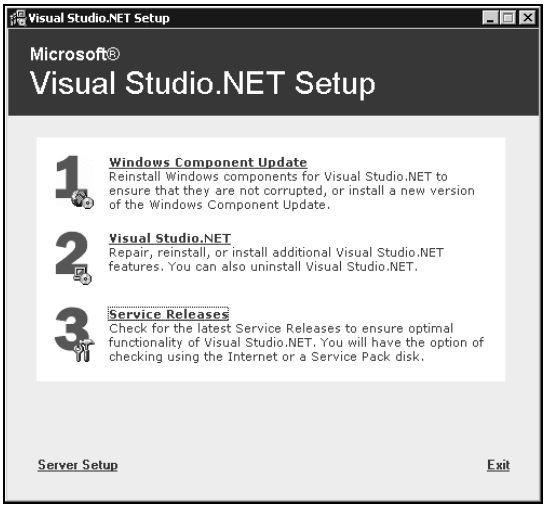
   As you can see there is quite an impact on your free disk space if you install all of the documentation!

9. Once you have chosen all of the features you want, click on Install Now! Installation will begin and you can sit back and relax for a bit. The setup time varies depending on how many features you chose to install. As a reference, the installation process took over an hour on a 650 MHz laptop computer with 256MB RAM, a 12GB hard drive, and running Windows 2000 Server. Keep in mind that this included all of Visual Studio .NET and all of the documentation.

10. When installation is completed, you will see the following:

Here you will see any problems that Setup encountered along the way. You are also given the chance to look at the **installation log**. This log provides a list of all actions taken during the installation process. Unless your installation reported errors, the installation log can safely be ignored. The Visual Studio .NET setup is nearly complete. Click on Done to move on to the final step.

11. We are returned to the initial setup screen again and the third choice is now available:



It is a good idea to select Service Releases to check for updates. Microsoft has done a good job of making software updates available through the Internet. These updates can include anything from additional documentation, to bug fixes. You will be given the choice to install any updates via a Service Pack CD or the Internet. Obviously, the Internet option requires an active connection. Since updates can be quite large, a fast connection is highly recommended.

Once you have performed the update process, Visual Basic .NET is ready to use. Now the real fun can begin! So get comfortable, relax, and let's enter the world of Visual Basic .NET.

The Visual Basic .NET IDE

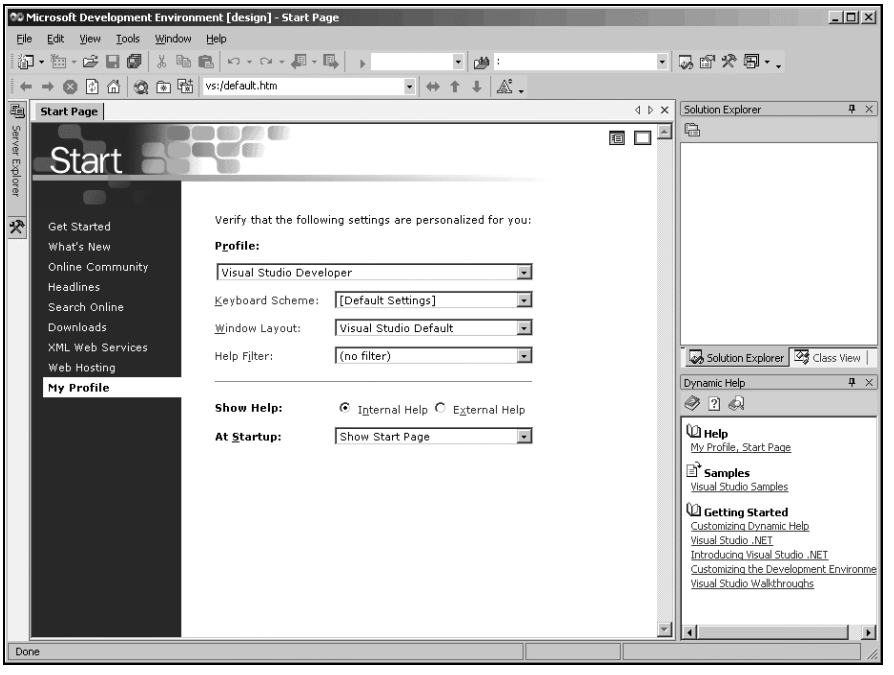
You don't actually need the Visual Basic .NET product to write applications in the Visual Basic .NET language. The actual ability to run Visual Basic .NET code is included with the .NET Framework. You could actually just write all of your Visual Basic .NET using a text editor such as Notepad.

However, by far the easiest way to write in Visual Basic .NET is by using the Visual Studio.NET **Integrated Development Environment**, also known as the **IDE**. This is what you actually see when working with Visual Basic .NET – the windows, boxes, etc. The IDE provides a wealth of features that are unavailable in ordinary text editors – such as code checking, visual representations of the finished application, and an explorer that displays all of the files that make up your project.

Let's look at the Visual Basic .NET IDE now!

The Profile Setup Page

An IDE is a way of bringing together a suite of tools that make developing software a lot easier. Let's fire up Visual Basic .NET and see what we've got. If you used the default installation, go to your Windows Start menu and then Programs | Microsoft Visual Studio.NET 7.0 | Microsoft Visual Studio.NET 7.0. A splash screen will briefly appear and then you should find yourself presented with the Start screen's My Profile tab:



This screen allows us to do some basic configuration of the IDE so that it serves us better. Since this IDE serves all the Visual Studio .NET languages, there are some settings to tailor it to our particular development interests.

Try It Out – Setting Up Our Profile

1. We are learning how to program using Visual Basic .NET, so select Visual Basic Developer from the drop-down box. The IDE will now rearrange itself (it actually looks similar to the Visual Basic 6 IDE).

2. You will also notice that the Keyboard Scheme and Window Layout options have changed to show: Visual Basic 6. If you are at all familiar with earlier versions of Visual Basic, then this should make you feel right at home. The Window Layout options rearrange the windows in the IDE to look similar to previous versions of other Microsoft development tools. How you lay out your windows in the future will be a matter of preference, but for now let's use the Visual Basic 6 option.

3. The Help Filter drop-down box also allows the help system to focus better on what you will find most useful. Set the Help Filter to Visual Basic.

4. The Show Help radio buttons allow us to select where help topics are displayed – Internal Help shows topics within the IDE window (the same window where you see the Start Page), whereas the External Help option opens topics in a separate window. This choice between Help displayed within the IDE or in a separate window is a matter of personal preference. However, until you are comfortable manipulating the various windows of the IDE, the external option might prove more useful, as the IDE remains constant while Help is open. You may receive a message that the change will not take effect until the next time you start Visual Studio .NET.

5. The At Startup pull-down permits you to define what you see whenever you first start Visual Studio .NET. The options here are:

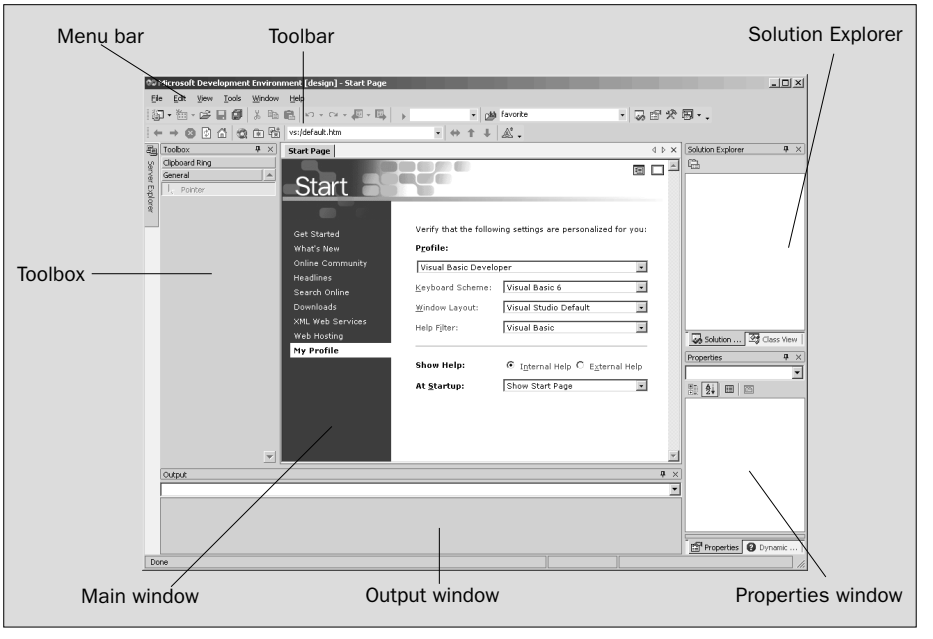
Option	Description
Show Start Page	Shows the Start Page (with its Get Started tab selected)
Load last loaded Solution	Opens last used solution
Show Open Project dialog box	Allows opening of a previously created project
Show New Project dialog box	Allows creation of a new project
Show Empty environment	Opens only the IDE

Note here that a *project* is a group of forms and code files that work together to create your application and these files are usually compiled into a single file. Sometimes when we create very complex applications, we need to group more than one project together in a *solution*. By default, however, each new project starts in its own separate solution.

6. Once you have configured your profile to your liking, select Get Started from the vertical menu bar to the left to begin using Visual Basic .NET.

The Get Started Page

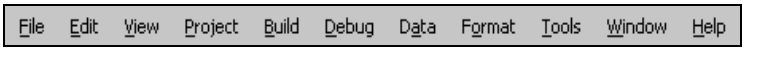
By now, you may be a bit anxious to start writing some code. First, let's take a brief look at the Get Started tab and see what is there. Assuming that you have been following along while setting up Visual Studio .NET, your screen should now look something like this:



Let's begin our exploration of the Visual Basic .NET IDE by looking at the toolbar and menu, which as you'll learn are not really that different from toolbars and menus you'll have seen in other Microsoft software such as Word, Excel, and PowerPoint.

The Menu

Visual Studio .NET's menu is dynamic, meaning that items will be added or removed depending on what you are trying to do. While we are still looking at the Get Started page, the menu bar will only consist of the File, Edit, View, Tools, Window, and Help menus. However, when you start working on a project, the full Visual Studio .NET menu appears as:



At this point, there is no need to cover each menu topic in great detail. You will become familiar with each as you progress through the book. Here is a quick rundown of what activities each menu item pertains to:

File

It seems every Windows program has a File menu. It has become the standard where you should find, if nothing else, a way to exit the application. In this case, you can also find ways of opening and closing single files and whole projects.

Edit

The Edit menu provides access to the items you would expect: Undo, Redo, Cut, Copy, Paste, and Delete.

View

The View menu provides quick access to the windows that make up the IDE, such as the Solution Explorer, Properties window, Output window, Toolbox, etc.

Project

The Project menu allows you to add various extra files to your application.

Build

The Build menu becomes important when you have completed your application and want to be able to run it without the use of the Visual Basic .NET environment (perhaps running it directly from your Windows Start menu as you would any other application such as Word or Access).

Debug

The Debug menu allows you to start and stop running your application within the Visual Basic .NET IDE. It also gives you access to the Visual Studio .NET **debugger**. The debugger allows you to step through your code while it is running to see how it is behaving.

Data

The Data menu helps you use information that comes from a database. It only appears when you are working with the visual part of your application (the [Design] tab will be the active one in the main window), not when you are writing code. Chapters 15 and 16 will introduce you to working with databases.

Format

The Format menu also only appears when you are working with the visual part of your application. Items on the Format menu allow you to manipulate how the windows you create will appear to the users of your application.

Tools

The Tools menu has commands to configure the Visual Studio .NET IDE, as well as links to other external tools that may have been installed.

Window

The Window menu has become standard for any application that allows more than one window to be open at a time, such as Word or Excel. The commands on this menu allow you to change the physical layout of the windows in the IDE.

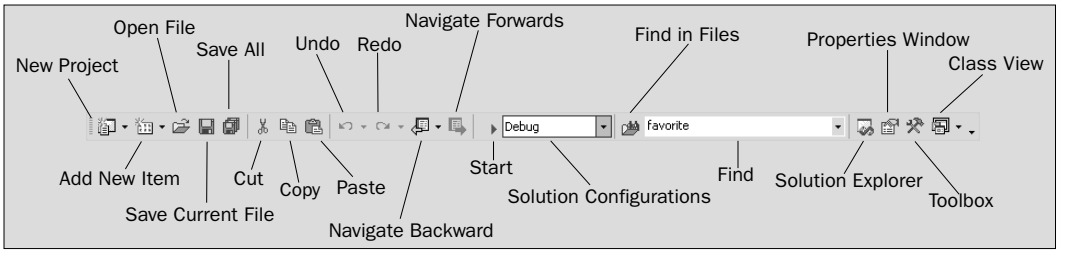
Help

The Help menu provides access to the Visual Studio .NET documentation. There are many different ways to access this information (for example, via the help contents, an index, or a search). The Help menu also has options that connect to the Microsoft Web site to obtain updates or report problems.

The Toolbars

There are many toolbars available within the IDE, including Formatting, Image Editor, and Text Editor, which you can add to and remove from the IDE via the View | Toolbars menu option. Each one provides quick access to often-used commands, preventing you from having to navigate through a series of menu options. For example, the leftmost icon on the toolbar shown below (New Project) is available from the menu by navigating to File | New | Project.

The default toolbar (called Standard) appears at the top of the IDE as:



The toolbar is segmented into groups of related options, which are separated by a vertical bar.

The first five icons provide access to the commonly used project and file manipulation options available through the File and Project menus, such as opening and saving files.

The next group of icons is for editing (Cut, Copy, and Paste).

The third group of icons is for editing and navigation. The navigation buttons replicate functionality found in the View menu and allow us to cycle through the tabs at the top of the main window.

The fourth group of icons provides the ability to start your application running (via the blue triangle) and to specify **build configurations**. There are times when you want certain parts of your code only to appear in a **debug version**, a bit like a rough draft version of your application. For example, you may have code in your application that is only useful for tracking down problems in the application. When it is time to release your application to the world, you will want to exclude this code by setting the Solution Configurations settings to Release. You can also access the functionality offered by this group via the Build and Debug menus.

The next section allows you to locate parts of your code quickly. The simplest way to search is to type some text into the Find textbox and hit *Enter*. If the text is found, it will be highlighted in the central window. The Find in Files option allows you to specify more sophisticated searches, including matching the case of the text, looking in specific files or projects, and replacing the found text with new text. The search functionality can also be accessed via the Edit | Find and Replace menu option.

The next group of icons provides quick links back to the Solution Explorer, Properties window, Toolbox, and Class View. If any of these windows are closed, clicking the appropriate icon will bring it back into view.

> **If you forget what a particular icon does, you can hover your mouse pointer over it so that a tooltip appears displaying the name of the toolbar option.**

We could continue to look at each of the other windows directly from the Start Page. But, as you can see they're all empty at this stage, and therefore not too revealing. The best way to look at the capabilities of the IDE is to use it while writing some code.

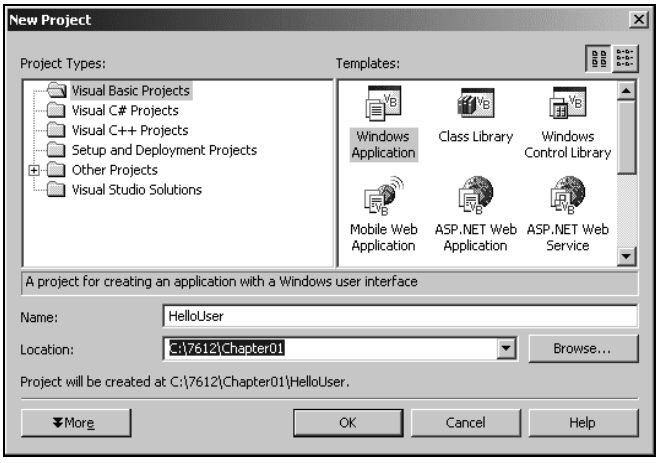
Creating a Simple Application

To finish our exploration of the Visual Basic .NET IDE we need to create a project, so that the windows labeled in the earlier diagram actually have some interesting content for us to look at. We're now going to create a very simple application called `HelloUser` that will allow us to enter a person's name and will display a greeting to that person in a message box.

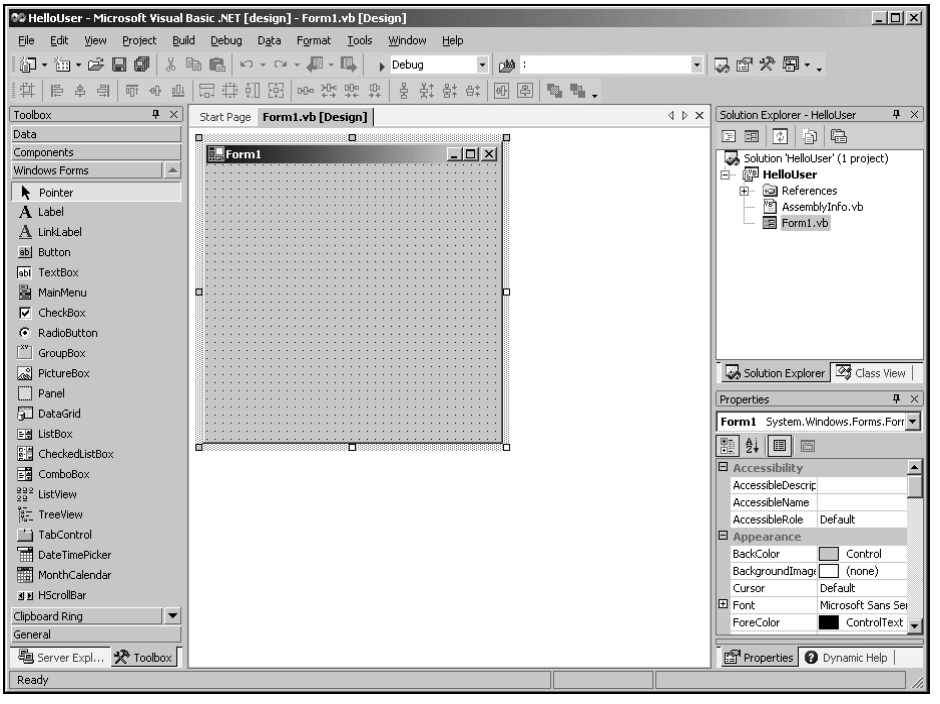
Try It Out – Creating a HelloUser Project

1. Click on the New Project button on the Start Page.

2. The New Project dialog box will open. Make sure you have Visual Basic Projects selected in the Project Types tree-view box to the left. Next, select Windows Application in the Templates box on the right. If you need to save this project to a location other than the default, be sure to enter it into the Location box. Finally, type HelloUser in the Name text box and click on the OK button:



3. Visual Basic .NET will then create an empty Windows application for us. So far, our HelloUser program consists of one blank window called a **Windows Form** (or sometimes just a **form**), with the default name of Form1.vb:



Whenever Visual Studio .NET creates a new file, either as part of the project creation process, or when you create a new file, it will use a name that describes what it is (in this case, a form) followed by a number.

Windows in the Visual Studio .NET IDE

At this point, you can see that the various windows in the IDE are beginning to show their purposes, and we will take a brief look at them now before we come back to the *Try It Out*. Note that if any of these windows are not visible on your screen, you can use the View menu to select and show them. Also if you do not like the location of any particular window you can move it by clicking on its title bar (the blue bar at the top) and dragging it to a new location. The windows in the IDE can **float** (stand out on their own) or be **dockable** (as they appear above).

Server Explorer

The Server Explorer gives you management access to the servers on your network. Here you can create database connections, and view the services provided by the available servers.

Toolbox

The Toolbox contains reusable components that can be inserted into your application. These can range from buttons to data connectors to customized controls either purchased or developed yourself.

Design Window

The Design window is where a lot of the action takes place. This is where you will draw your user interface and write your code. This window is sometimes referred to as the **Designer**.

Solution Explorer

The Solution Explorer window contains a hierarchical view of your solution. A solution can contain many projects while a project contains code and code references that solve a particular problem.

Class View

The Class View window (shown as a tab with the Solution Explorer) gives you a tree view of the classes in your program and shows the properties and methods that each contains. A **class** is code file that groups data and the functions that manipulate it together into one unit. A **property** is data, and a **method** is a function or subroutine.

Properties

The Properties window shows what properties the selected object makes available. Although you can set these properties in your code, sometimes it is much easier to set them while you are designing your application. You will notice that the File Name property has the value Form1.vb. This is the physical file name for the form's code and layout information.

Task List

The Task List window highlights any errors encountered when you try to run your code. Clicking on the item in this window will take you to the line of code containing the error.

Output

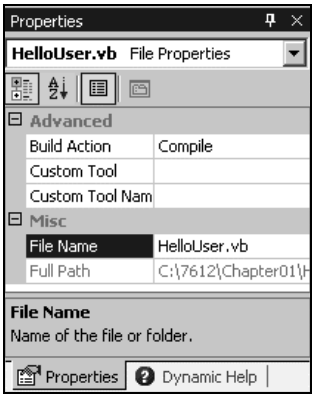
When you run your code the progress made in reading it (or **compiling** it) is registered via messages posted in the Output window.

Dynamic Help

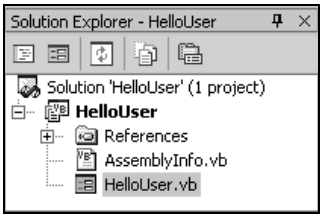
The Dynamic Help window displays a list of help topics that relate to whatever in the IDE has focus. If you click on the form in the Design Window and then open Dynamic Help, you will see a list of help topics relating to forms.

Try It Out – Creating a HelloUser Project – Continued

1. Let's change the name of our form to something more indicative of what our application is. Click on Form1.vb in the Solution Explorer window. Then, in the Properties window, change the File Name property from Form1.vb to HelloUser.vb and hit *Enter*: When changing properties you must either hit *Enter* or click off the property for it to take effect.

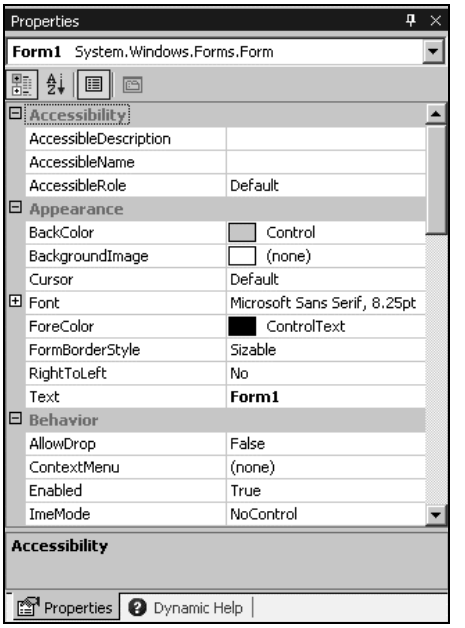


2. Notice that the form's file name has also been updated in the Solution Explorer to read HelloUser.vb:



3. Now click on the form displayed in the main window; the Properties window will change to display the form's **Form properties** (instead of the **File properties**, which we have just been looking at). You will notice that the Properties window is dramatically different. The difference is the result of two different views of the same file. When the form name is highlighted in the Solution Explorer window, the physical file properties of the form are displayed. When the form in the Design View is highlighted, the visual properties and logical properties of the form are displayed.

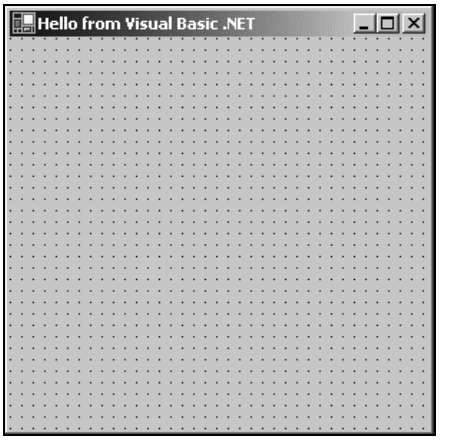
The Properties window allows us to easily set a control's properties. Remember properties are a particular object's set of internal data. Properties usually describe appearance or behavior. In the screenshot, you can see that properties are grouped together in categories – Accessibility, Appearance, and Behavior are the ones shown in here:



You can see that under the Appearance category are such properties as BackColor (the form's background color), Font (the typeface used for text on the form), and Text (the form's caption displayed in the title bar). These are all examples of properties that describe the form's appearance.

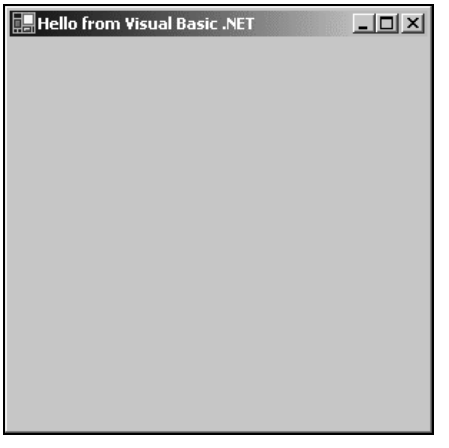
One property, FormBorderStyle, tells Windows how to draw the form. In this case, the window is resizable, meaning that when the form is displayed, the user can change its size (like most application windows).

4. Right now, the title of our form (displayed in the bar at the top) is Form1. This isn't very descriptive, so let's change it to reflect the purpose of this application. Locate the Text property in the Appearance section of the Properties window and change its value to Hello from Visual Basic .NET and hit *Enter*. Notice that the form's title has been updated to reflect the change:



If you have trouble finding properties, click the little AZ button on the toolbar towards the top of the Properties window. This changes the property listing from being ordered by category to being ordered by name:

5. We are now finished. Click on the Start button on the Visual Studio .NET toolbar (the blue triangle) to run the application. As you work through the book, whenever we say "run the project" or "start the project", just click on the Start button. An empty window with the title Hello from Visual Basic .NET is displayed:



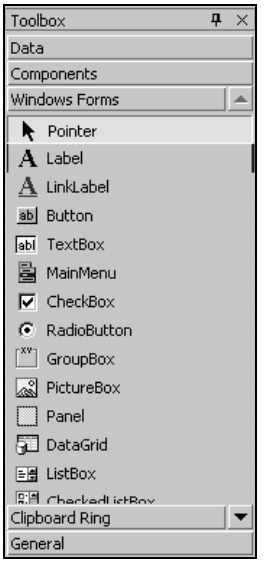
Notice how the grid pattern of dots has now disappeared. These are displayed at design time to help us place controls such as boxes, labels, and radio buttons onto our form. They're not needed (or even particularly desirable) at run time, so they're not shown.

OK, that was simple, but our little application isn't doing much at the moment. Let's make it a little more interactive. To do this we are going to add some controls – a label, textbox, and two buttons to the form. This will let us see how the toolbox makes adding functionality quite simple. You may be wondering at this point when we will actually look at some code. Soon! The great thing about Visual Basic .NET is that you can develop a fair amount of your application *without* writing any code. Sure, the code is still there, behind the scenes, but as we'll see, Visual Basic .NET writes a lot of it for us.

The Toolbox

The Toolbox is accessed via the View | Toolbox menu option, the Toolbox icon on the Standard menu bar, or by pressing *Ctrl + Alt + X*.

The Toolbox contains a tabbed view of the various controls and components that can be placed onto your form. Controls such as textboxes, buttons, radio buttons, and drop-down boxes can be selected and then *drawn* onto your form. For the `HelloUser` application, we will only be using the controls on the Windows Forms tab:



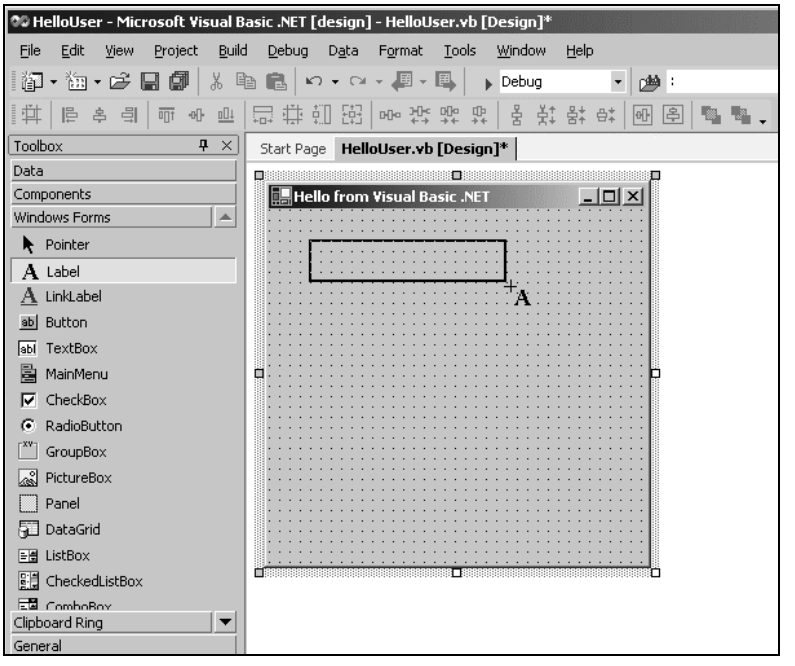
Here we can see a listing of standard .NET controls for Windows forms. The down arrow button to the right of the Clipboard Ring tab title actually scrolls the Windows Forms control list down as there are too many to fit in otherwise. The up arrow button on the Windows Forms tab scrolls the list up. Note that the order in which your controls appear may be different.

Controls can be added to your forms in any order, so it does not matter if we add the label control after the textbox or the buttons before the label.

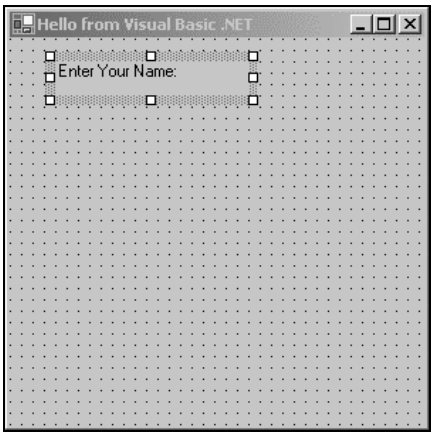
Try It Out – Adding Controls to the HelloUser Application

1. Stop the project if it's still running, as we now want to add some controls to our form. The simplest way to do this is to click on the X button in the top right corner of the form. Alternatively, you can click on the blue square in the Visual Studio .NET IDE (which displays the text Stop Debugging if you hover over it with your mouse pointer).

2. Let's add a Label control to the form. Click on Label in the Toolbox to select it. Move the cursor over the form's Designer. You'll notice that the cursor looks like a crosshair with a little floating letter A beneath it. Click and hold the mouse button where you want the top left corner of the label and drag the mouse to where you want the bottom right. (Placing controls can also be accomplished by double-clicking on the required control in the Toolbox.)



3. If the Label control you have just drawn is not in the your desired location, or is too big or too small, that is not really a problem. Once the control is on the form you can resize it or move it around. The image opposite shows what the control looks like after you place it on the form. To move it, click on the gray dotted border and drag it to the desired location. To resize it, click and drag on one of the white box "handles" and stretch the control in the needed direction. The corner handles resize both the horizontal and vertical dimension at the same time:



4. After drawing a control on the form, we should at least configure its name and the text that it will display. You'll see that the Properties window to the right of the Designer has changed to Label1, telling you that you're currently examining the properties for it. In the Properties window, set our new label's Text property to Enter Your Name: and its (Name) property to lblName:

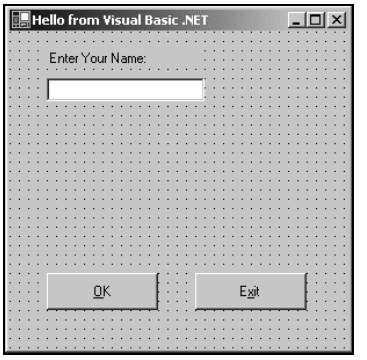
5. Now, directly beneath the label, we want to add a textbox, so that we can enter a name. We're going to repeat the procedure we followed for adding the label, but this time make sure you select the TextBox from the toolbar. Once you have dragged-and-dropped (or double-clicked) the control into the appropriate position, use the Properties window to set its Name property to txtName and clear the Text property so that the textbox now appears to be blank.

Notice how, out of the eight sizing handles surrounding the control, only two are shown in white. By default, the TextBox control cannot be made any taller than the absolute height necessary to contain the font that it will use to draw the text.

6. In the bottom left corner of the form, add a Button control in exactly the same manner as you added the label and textbox. Set its Name property to OK, and its Text property to &OK.

The ampersand (&) is used in the Text property of buttons to create a keyboard shortcut (known as a *hot key*). The letter with the & sign placed in front of it will become underlined to signal to users that they can select that button by pressing the *Alt-letter* key combination, instead of using the mouse (on some configurations the underline doesn't appear to the user until they press ALT). In this particular instance, pressing *Alt+O* would be the same as clicking directly on the OK button. There is no need to write code to accomplish this.

7. Now add a second Button control to the bottom right corner of the form and set the Name property to btnExit and the Text property to E&xit. Your form should look similar to this:



Modified Hungarian Notation

You may have noticed that the names given to the controls look a little funny. Each name is prefixed with a shorthand identifier describing the type of control it is. This makes it much easier to understand what type of control we are working with when we are looking through code. For example, say we had a control called simply `Name`, without a prefix of `lbl` or `txt`, we would not know whether we were working with a textbox that accepted a name or a label that displayed a name. Imagine if, in the previous *Try It Out*, we had named our label `Name1` and our textbox `Name2` – we'd very quickly become confused. How about if we left our application for a month or two, and then came back to it to make some changes?

When working with other developers, it is very important to keep the coding style consistent. One of the most commonly used styles used for controls within application development in many languages is **Modified Hungarian notation**. The notion of prefixing control names to identify their use was brought forth by Dr. Charles Simonyi. He worked for the Xerox Palo Alto Research Center (XPARC), before joining Microsoft. He came up with short prefix mnemonics that allowed programmers to easily identify the type of information a variable might contain. Since Dr. Simonyi is Hungarian, and the prefixes make the names look a little foreign, the name Hungarian Notation stuck. Since the original notation was used in C/C++ development, the notation for Visual Basic .NET is termed Modified. Here is a table of some of the commonly used prefixes that we shall be using in this book:

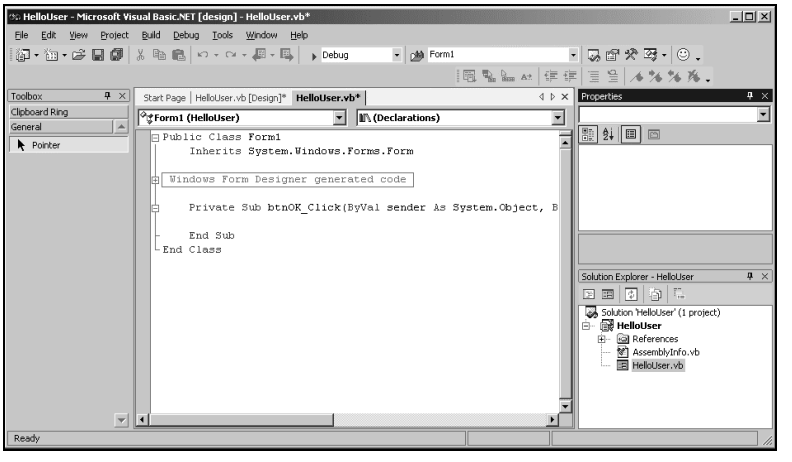
Control	Prefix
Button	`btn`
ComboBox	`cbo`
CheckBox	`chk`
Label	`lbl`
ListBox	`lst`
MainMenu	`mnu`
RadioButton	`rdb`
PictureBox	`pic`
TextBox	`txt`

Hungarian Notation can be a real time-saver when looking at code someone else wrote, or at code that you have written months past. However, by far the most important thing is to be consistent in your naming. When you start coding, pick a convention for your naming. It is recommended that you use the *de facto* standard Modified-Hungarian for Visual Basic .NET, but it is not required. Once you pick a convention, stick to it. When modifying someone else's code, use theirs. There is very little code that is ever written, put into production and then forgotten. A standard naming convention followed throughout a project will save countless hours when the application is maintained.

Now let's get back to the application. It is now time to write some actual code.

The Code Editor

Now that we have the HelloUser form defined, we have to add some code to actually make it do something interesting. We have already seen how easy it is to add controls to a form. Providing the functionality behind those on-screen elements is not much more difficult. To add the code for a control, just double-click on it. This will open the code editor in the main window:



Notice that an additional tab has been created in the main window. Now we have the Design tab and the code tab. We drew the controls on the Design tab, and we write code on the code tab. One thing to note here is that we have not created a separate file for the code. The visual definition and the code behind it both exist in the same file: `HelloUser.vb`. This is actually the reason why building applications with Visual Basic .NET is so slick and easy. Using the Design view you can visually lay out your application, and then using the Code view add just the bits of code to implement your desired functionality.

You will also notice that there are two pull-downs at the top of the window. These provide shortcuts to the various parts of our code. If you pull down the one on the left, Form1 (HelloUser), you will see a list of all of the objects within our application. If you pull down the one on the right, (Declarations), you will see a list of all of the defined functions or subroutines. If this particular form had a lot of code behind it, these pull-downs would make navigating to the desired area very quick – jumping to the selected area. However, since all of the code fits in the window, there are not a lot of places to get lost.

Now let's look at the code in the window. The code in Visual Studio .NET is set up into **regions** designated by the plus (+) and minus (-) buttons along the left side. These regions can be collapsed and expanded in order to simplify what you are looking at. If you expand the region labeled `Windows Form Designer generated code`, you will see a lot of code that Visual Basic .NET has automatically generated for us, which takes care of defining each of the controls on the form and how the form itself should behave. We do not have to worry about the code in this region, so collapse the `Windows Form Designer generated code` region once more and let's just concentrate on the code we have to write ourselves.

Try It Out – Adding Code to the HelloUser Project

1. To begin adding the necessary code, click on the Design tab to show the form again. Then double-click on the OK button. The code window will reopen with the following code. This is the shell of button's `Click` event and is the place where we enter code that we want to be run when we click on the button. This code is known as an **event handler**, and sometimes is also referred to as an **event procedure**:

```
Private Sub OK_Click(ByVal sender As SystemObject, ByVal e As _
            System.EventArgs) Handles OK_Click

End Sub
```

> **Due to the typographic constraints in publishing, it is not possible to put the `Sub` declaration on one line. Visual Basic .NET allows you to break up lines of code by using the underscore character (_) to signify a line continuation. The space before the underscore is required. Any whitespace preceding the code on the following line is ignored.**

`Sub` is an example of a **keyword**. In programming terms, a keyword is a special word that is used to tell Visual Basic .NET to do something special. In this case, it tells Visual Basic .NET that this is a procedure. Anything that we type between the lines `Private Sub` and `End Sub` will make up the event procedure for the OK button.

2. Now add the highlighted code into the procedure:

```
Private Sub OK_Click(ByVal sender As SystemObject, ByVal e As _
            System.EventArgs) Handles OK_Click
    'Display a message box greeting the user
    MessageBox.Show("Hello, " & txtName.Text & _
                    "! Welcome to Visual Basic .NET.", _
                    "HelloUser Message")
End Sub
```

Throughout this book, you'll be presented with code that you should enter into your program if you're following along. Usually, we'll make it pretty obvious where you put the code, but as we go we'll explain anything that looks out of the ordinary.

3. After you have added the code, go back to the Design tab, and double-click on the Exit button. Add the highlighted code to the `ExitButton_Click` event procedure.

```
Private Sub ExitButton_Click(ByVal sender As SystemObject, ByVal e As _
            System.EventArgs) Handles ExitButton_Click
    'End the program
    Me.Dispose()
End Sub
```

You may be wondering what `Me` is. `Me` refers to the form. Just like the pronoun, *me*, it is just a shorthand for referring to oneself.

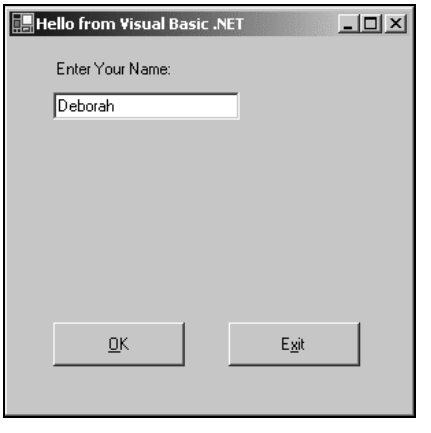
4. Now that the code is finished, the moment of truth has arrived and we can see our creation. First though, save your work by using File | Save from the menu, or by clicking the disk icon on the toolbar.

5. Now click on the Start button on the toolbar. You will notice a lot of activity in the Output window at the bottom of your screen. Providing you haven't made any mistakes in entering the code, this information just lets you know what files are being loaded to run your application.

   It's at this point that Visual Studio .NET will **compile** the code. Compiling is the activity of taking the Visual Basic .NET source code that you've written and translating it into a form that the computer understands. After the compilation is complete, Visual Studio .NET will **run** (also known as **execute**) the program and we'll be able to see the results.

If Visual Basic .NET encountered any errors, they will be displayed as tasks in the Task List window. Double-clicking on a task will transport you to the offending line of code. We will learn more about how to debug the errors in our code in Chapter 11.

6. When the application loads you will see the main form. Enter a name and click on OK (or press the *Alt+O* key combination):



7. A window known as a message box appears, welcoming the person whose name was entered in the textbox to Visual Basic .NET – in this case Deborah:

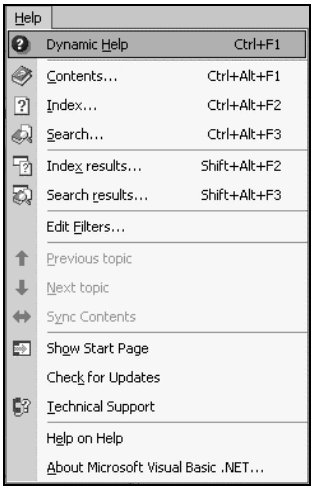
8. After you close the message box by clicking on its OK button, click on the Exit button on our form. The application will close and you will be brought back to the Visual Basic .NET IDE.

   That completes our first Visual Basic .NET application. Well done! Before we move on to the next chapter, let's take a quick look at the Visual Studio .NET Help system.

Using the Help System

The **Help system** included with Visual Basic .NET is an improvement over Help systems in previous versions. As you begin to learn Visual Basic .NET, you will probably become very familiar with the Help system. However, it is worthwhile to give you an overview, just to help speed your searches for information.

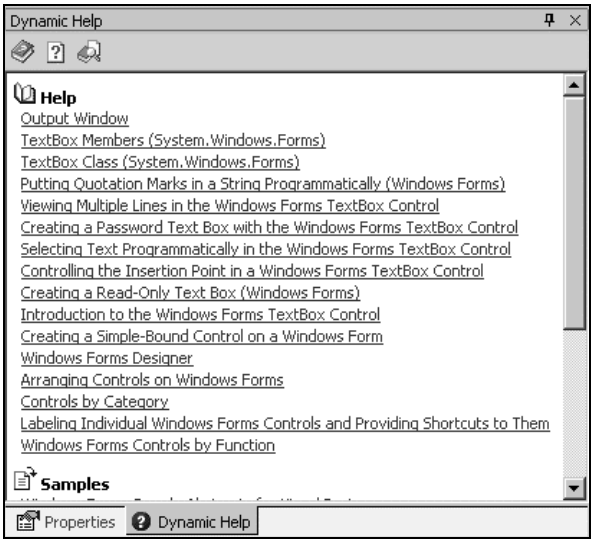
The Help menu appears as:



As you can see this menu contains many more entries than the typical Windows application. The main reason for this is the vastness of the documentation. Few people could keep it all in their heads – but luckily, that's not a problem, as we can always quickly and easily refer to the Help system. Think of it as a safety net for your brain.

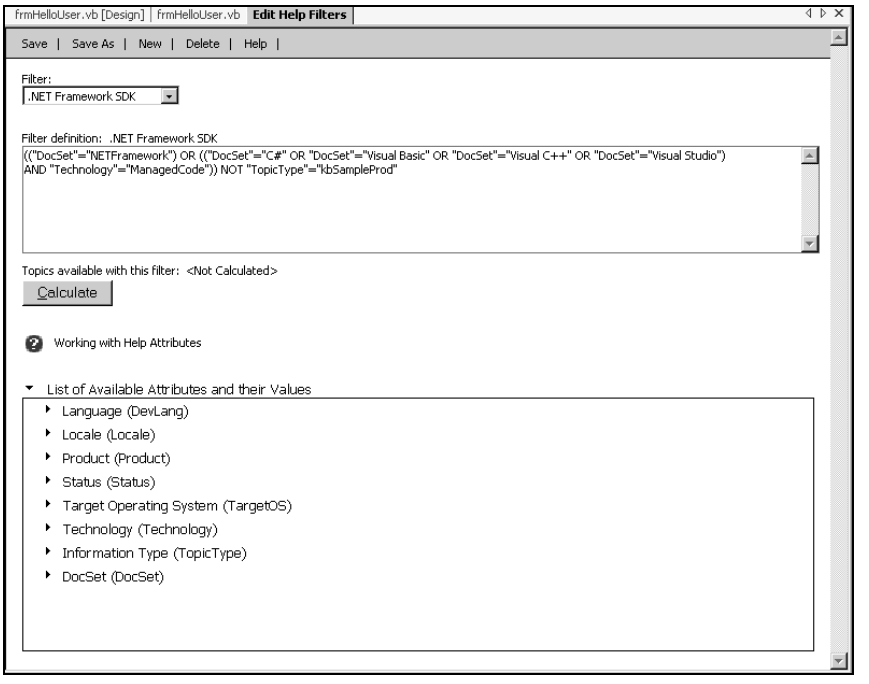
One really fantastic new feature is Dynamic Help. If you turn this option on (by selecting Dynamic Help from the Help menu), a window will display a list of relevant topics for whatever you may be working on. If you followed the default installation and have not rearranged the IDE, the Dynamic Help is displayed as a tab behind Properties.

Let's say for example, that we are working with a textbox (perhaps the textbox in the `HelloUser` application) and want to find out some information; we just select the textbox on our form and we can see all the help topics that pertain to textboxes:



The other help commands in the Help menu (Contents, Index, and Search), function just as they would in any other Windows application.

The domain of help topics to search and display is defined by the profile that we defined at installation. However, the Edit Filters… menu option allows you to further focus what types of documentation to include in the search:



The filter is a logical statement that either includes or excludes documentation, based on some predefined types. The statement is in the upper text area; the document attributes and values are listed below. Although the filter equation can be edited directly, selecting attributes from the list automatically updates the statement. You can then save your changes to the current filter name, or use Save As to create a new one based on the existing equation.

Summary

Hopefully, you are beginning to see that developing basic applications with Visual Basic .NET is not that difficult. We have taken a look at the IDE and how it can help you put together good software quickly. The Toolbox allows you to add controls quickly and easily to your programs. The Properties window makes configuring those controls a snap, while the Solution Explorer gives us a bird's eye view of the files that make up our project. We even wrote a little code.

Questions

At this point, we have not covered much about Visual Basic .NET, the language. So these exercises won't be too difficult.

1. What Modified-Hungarian prefix should you use for a combo box? A label? A textbox?

2. (*This assumes you set the* Help Filter *to* Visual Basic and Related.) Open the Help System and search for MessageBox. Notice how many topics are returned. Change the Help Filter option on the My Profile screen, to No Filter. Repeat the search for MessageBox. Did the Help System return more or fewer topics?

3. When creating a button, how would you make the button respond to a keyboard hot key?

The answers for these questions and those at the end of all the other chapters can be found in Appendix B.