CHAPTER 1

Visual Studio .NET

at the completion of this chapter, you will be able to . . .

1. Distinguish the features of the Visual Studio IDE versus the .NET Framework.

2. Identify and understand the purpose of each of the files listed in the Solution Explorer.

3. Understand what happens at compile time.

4. Display data from the assembly attributes in AssemblyInfo.vb.

5. Recognize features from the VB6 compatibility library and why you should avoid them.

6. Create an MDI project with a parent form, child forms, a toolbar, status bar, context menu, and ToolTips.
Microsoft has revolutionized the programming for Windows applications and has become a bigger player in the development of Web applications with the introduction of the .NET Framework and Visual Studio .NET. These new products introduce significant changes into program development for Visual Basic. Not only does .NET bring true object-orientation to the language, it also provides great advances in the ease of developing projects for cross-platform compatibility.

Two major parts of .NET are the Microsoft .NET Framework Software Development Kit (SDK) and the Visual Studio Integrated Development Environment (IDE). The IDE is used to develop programs while the Framework runs the programs.

The .NET Framework

The .NET Framework provides a platform for developing and running applications and XML Web Services written in multiple languages on multiple platforms. The Framework is composed of the common language runtime, class libraries, and ASP.NET—a component-based version of active server pages (ASP).

Common Language Runtime

The common language runtime (CLR) is an environment that manages execution of code. It provides services for tasks such as integrating components developed in different languages, handling errors across languages, handling security, and managing the storage and destruction of objects.

Any code that is compiled to run in the CLR is called managed code. The managed code automatically contains metadata, which means data that describe data. A common language runtime portable executable (PE) file contains the metadata along with the code. The metadata include data types, members, references, and information needed to load classes and to call methods from a class.

The CLR also manages data storage. Objects that are no longer being used are automatically removed from memory by the garbage collector component of the CLR. When you allow the runtime to handle the garbage collection of objects, the data are referred to as managed data. Although you can manage the memory of your data, it is usually better to let the runtime handle it.

Your code can be integrated with classes and methods of managed code written in other programming languages. The CLR has standards for data types allowing you to pass an instance of one of your classes to a method created in a different language. Although we will not be doing any cross-language programming in this text, you should be aware of this powerful feature.

Class Library

All of the classes and interfaces that are a part of the .NET language are stored in a library known as the .NET Framework class library. The library is divided into sections or groups known as namespaces. You should be familiar with some of the common namespaces such as System and System.Drawing. Each namespace contains classes, structures, enumerations, delegates, and/or inter-
faces that you can use in your programs. Table 1.1 shows some of the namespaces in the .NET Framework class library.

The classes in the library comply with published standards known as the **Common Language Specification (CLS)**. The CLS specifies how a language that interacts with the CLR should behave. If you want your programs to interact with programs and components written in other languages, you should make sure that they are CLS-compliant. The rules for CLS compliance can be found in the .NET Framework Developer’s Guide under the heading “What Is the Common Language Specification?”

**Types**
The .NET documentation uses the general term *types* to refer to the classes, structures, enumerations, delegates, interfaces, and data types in the library, as well as any that you define. You can think of a type as any element that you can use in the `As` clause of a declaration:

```vbnet
Dim AnyName As SomeType
```

**Value Types versus Reference Types**
When you declare a variable, it may be considered a *value type* or a *reference type*. The difference between the two determines how the runtime will treat the variables when you assign one variable to another. For example, if you assign one integer variable to another, you have two memory locations with the same value:

```vbnet
intSecondValue = intFirstValue
```
However, if you assign one reference type to another, you have two variables that point to the same object in memory:

```vbnet
frmSecondForm = frmFirstForm  'Assign reference for first form to second form
```

Any changes that you make to either variable are made to the one object in memory to which both variables refer. In previous versions of VB, reference types were called object variables. However, VB .NET reference types include more types than the object variables of VB 6.

```vbnet
'Value types
Dim intFirstValue As Integer = 10
Dim intSecondValue As Integer
intSecondValue = intFirstValue
intFirstValue = 5
Debug.WriteLine("intFirstValue = " & intFirstValue.ToString() & 
    "; intSecondValue = " & intSecondValue.ToString())

'Reference types
Dim frmFirstForm As New Form1()
Dim frmSecondForm As Form1
frmSecondForm = frmFirstForm  'Assign reference for first form to second form
frmSecondForm.Text = "Second Form Caption"
frmFirstForm.Text = "New Caption for First Form"
'What is the Text property of frmFirstForm? Of frmSecondForm?
Debug.WriteLine("frmFirstForm = " & frmFirstForm.Text & 
    "; frmSecondForm = " & frmSecondForm.Text)
```

All numeric data types are value types. Reference types include class types, arrays (even if the individual elements are numeric), and strings. A value type always holds a value; when you declare a new variable of a value type, the variable is always initialized, either to a value that you supply or to the default value. A reference type may or may not hold a value; you can use the IsNothing function to determine whether the variable refers to an actual object.

### Compiling to Intermediate Language

The program code that you write is referred to as source code. The compiler translates your code into Microsoft intermediate language (MSIL) or sometimes referred to as just IL. MSIL is a platform-independent set of instructions that is combined with the metadata to form a file called a portable executable (PE) file, which has an .exe or .dll extension. When your program runs, the MSIL is converted to the native code of the specific machine using a just-in-time (JIT) compiler, which is part of the CLR (Figure 1.1).

**Figure 1.1**
The steps in compiling and executing a program using the .NET CLR.
Assemblies

An assembly is a basic unit of code that may be a single PE file or multiple files. Each of your applications will be contained in a single assembly, which is the smallest deployable piece of code. An assembly has an assembly manifest that contains metadata about the version, a table describing all of the files needed by the assembly, and an “assembly reference list” that specifies all of the external files needed, such as DLLs created by someone else. An assembly is similar to an exe or a dll file written in earlier versions of Visual Basic—it contains all of the necessary information to run the application or component.

Recall that the .NET Framework class library is not one big file but rather a collection of files. The classes are stored in many files with the extension .dll and referred to as DLLs (for dynamic link libraries). Each of the DLLs in the class library is one assembly. As your program begins execution, only the needed assemblies are loaded into memory. When you want to use a type that is not already referenced, you must add a reference to the DLL (assembly).

Attributes

Attributes are tags containing information about parts of a program such as types or methods. The system defines many attributes of your assembly, such as the name, version, culture, and security. The attributes are part of the metadata in a .NET assembly. The process of examining the metadata in an assembly’s attributes is called reflection.

Later in this chapter you will learn to retrieve and use the custom attributes in a project’s AssemblyInfo file.

References Collection

A reference object is used to connect a Visual Basic project to external components. The two types of reference objects are assemblies and COM objects. A reference to another project is an assembly reference and is called a project-to-project reference. COM objects are components written in previous versions of VB or other non-CLS-compliant languages.

ASP.NET

Another big part of the .NET world is the improvement in Web development. ASP.NET is the newest version of Active Server Pages (ASP). It is a Web development environment that can compile applications written in any .NET-compatible language including Visual Basic. This means that the benefits of the common language runtime and managed code are available for developing Web applications. ASP.NET makes Web development easier by providing the same debugging support for Web Forms and Web Services as for Windows applications.

You will begin working with ASP.NET in Chapter 6. Chapters 7 and 8 cover accessing databases from ASP.NET, which is a common technique for displaying data on a Web site.
Feedback 1.1

1. What is meant by the term .NET Framework?
2. What is the meaning and function of each of these terms?
   a. CLR  
   b. CLS  
   c. MSIL  
   d. PE

Visual Studio .NET

Although you could write your programs in any editor and then use the SDK to compile them, Visual Studio provides an environment to make your development task easier. You should already be familiar with the various windows in the environment as well as the basic debugging capabilities. For a review of the VS IDE, see Appendix E. This section introduces you to more details about the parts of a project.

Solution Explorer Files

Take a look at the files in the Solution Explorer for a Windows application (Figure 1.2). The files that you start with depend on the type of the project. When you click on the Show All Files button, you can see the hidden files and folders. You can see the References collection, the bin folder, the obj folder, the AssemblyInfo.vb file, the Form1.vb file, and the Form1.resx file. Expand each of the nodes to see more details. Notice that the obj folder contains a temporary PE folder. All of your compiled exe files are stored in the bin folder when you compile a project.

The Bin Folder

When your program compiles without errors—a clean compile—the resulting .exe or .dll file is stored in the bin folder. Notice in Figure 1.2 that the bin...
folder holds an .exe file and a .pdb file. The .exe or .dll file is the executable, which is used to run your program. The .pdb file holds debugging information. If you are distributing your application, you only need to distribute the .exe or .dll file, not the .pdb file.

If your program accesses the Application.StartupPath property, perhaps to store or retrieve a saved text file, the StartupPath points to the bin folder.

References
In the References folder you can see a list of System libraries (DLLs) such as System, System.Data, System.Drawing, System.Windows.Forms, and System.XML. You can add a reference by right-clicking on the References folder and selecting Add Reference from the context menu. You may need to add a reference if you want to refer to objects in another assembly or components written in a different language.

The Form’s .resx File
Each form in your project has a resource file with a .resx extension and the same name as the form file. For example, Form1.vb has a Form1.resx file, which you can see in the Solution Explorer when you click on the Show All Files button. If you rename your form file in the Solution Explorer, the .resx file is automatically renamed to match. At times you can get into trouble if you have already compiled the program and the file is not automatically renamed, or if a new .resx file is created by the IDE. You can safely delete any extra .resx files if their names do not match any forms in the project.

The .resx file is mostly text in XML format, which you can open and view in the IDE. Any graphic elements that you add to the form, such as a PictureBox’s Image property or a Form’s Icon property, are also stored in the .resx file in text that represents the binary graphic file. It’s the .resx file that supplies the graphics to the form at compile time.

AssemblyInfo.vb
Each project has an AssemblyInfo.vb file that contains many attributes with information about your application. This is the standard file that is generated by default:

```vbnet
Imports System.Reflection
Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following set of attributes. Change these attribute values to modify the information associated with an assembly.

' Review the values of the assembly attributes
<Assembly: AssemblyTitle("")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("")>
<Assembly: AssemblyCopyright("")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>```
'The following GUID is for the ID of the typelib if this project is exposed to COM
<Assembly: Guid("919A14A8-2540-4EDA-BAB6-9183818A0774")>

'Version information for an assembly consists of the following four values:
  '  Major Version
  '  Minor Version
  '  Build Number
  '  Revision

'You can specify all the values or you can default the Build and Revision Numbers
'by using the '*' as shown below:
<Assembly: AssemblyVersion("1.0.*")>

Notice the attributes for the version, the company, the product, the copyright, and the trademark. You can give these attributes values at design time and retrieve them at run time. You might want to retrieve and display the attribute values on a splash screen or an About form.

For the version attribute, you can enter any value you wish. By default, Visual Studio sets the version to “1.0.*”, which means that you want the compiler to automatically increment the version. For example, increment from 1.0.1 to 1.0.2 and then 1.0.3.

**Viewing a Project's Attributes** After you enter custom attributes in the AssemblyInfo.vb file and compile your project, you can view the attributes in Windows Explorer. Point to the filename in the project’s bin folder, either an .exe or .dll file, right-click, and choose Properties from the context menu. Display the Version tab to see the attributes (Figure 1.3). In Windows XP it's even easier: Point to the filename and pause; the attributes pop up automatically.

*Figure 1.3* Display the application’s attributes in Windows Explorer.
Retrieving Attributes To retrieve assembly attributes in a VB program, you should first import the System.Reflection namespace. Then you can declare an Assembly object. Note that Assembly is a reserved word; you must use brackets around the word to use Assembly as a class name, but the editor adds the brackets for you in a Dim statement.

```vbnet
'VB program code
Imports System.Reflection 'Place this statement before the class header

'Declare the variable and instantiate it in a procedure
Dim asmInfo As [Assembly] 'Retrieve the assembly information for this project
asmInfo = [Assembly].Load("ProjectName") 'Where ProjectName is the name of this project

Note that ProjectName is actually the name of your assembly, which by default is the same as the name of your project. But if you rename your project, the assembly name does not automatically change to the new name. You can change the project's assembly name in the Project Properties dialog box.

You can display the FullName property of an assembly, which gives information about the version:

```vbnet
dlblInfo.Text = asmInfo.FullName
```

The preceding line of code produces this string: "ProjectName, Version = 1.0.891.19636, Culture = neutral, PublicKeyToken = null"

Retrieving Custom Attributes Most of the attributes in the AssemblyInfo.vb file are considered custom attributes. You can retrieve attributes such as Title and Description by setting up an array of objects and using the GetCustomAttributes method of the Attribute object.

```vbnet
Dim objAttributes() As Object 'Declare an array of objects
objAttributes = asmInfo.GetCustomAttributes(False) 'Argument is ignored. Use true or false

To display an array element, you must declare an object of the correct attribute class and assign the corresponding array element to the new object. For example, to retrieve the Product attribute, you declare an AssemblyProductAttribute object.
```

```vbnet
Dim atrProduct As AssemblyProductAttribute
atrProduct = CType(objAssembly(n), AssemblyProductAttribute)
```

You can find the complete list of attribute types in MSDN under "System.Reflection." This is a partial list of some of the more useful attributes:

- AssemblyCompanyAttribute
- AssemblyCopyrightAttribute
- AssemblyDescriptionAttribute
- AssemblyProductAttribute
- AssemblyTitleAttribute
If Option Strict is on, you must convert the array element to an object of the appropriate type. And the sequence of items in the array can vary, so you must check the type of each array element to determine its attribute type. Use a For Each loop and a Select Case to retrieve the attributes you want.

```
Dim objItem As Object
'Convert from object to desired types
'Need Select Case because order of attributes in array varies
For Each objItem In objAttributes
    Select Case objItem.GetType.ToString()
    Case "System.Reflection.AssemblyTitleAttribute"
        atrTitle = CType(objItem, AssemblyTitleAttribute)
        lblTitle.Text = atrTitle.Title.ToString()
    Case "System.Reflection.AssemblyCompanyAttribute"
        atrCompany = CType(objItem, AssemblyCompanyAttribute)
        lblCompany.Text = atrCompany.Company.ToString()
    Case "System.Reflection.AssemblyProductAttribute"
        atrProduct = CType(objItem, AssemblyProductAttribute)
        lblProduct.Text = "We Sell: " & atrProduct.Product.ToString()
    Case "System.Reflection.AssemblyCopyrightAttribute"
        atrCopyright = CType(objItem, AssemblyCopyrightAttribute)
        lblCopyright.Text = atrCopyright.Copyright.ToString()
    End Select
Next
```

Test It Now it's time to try implementing some of these techniques by writing a program that displays attributes from the AssemblyInfo.vb file on a form.

Open a new Project in Visual Basic and open the default AssemblyInfo.vb file. Modify the code by inserting the following text:

```
<Assembly: AssemblyTitle("Attributes Demo")>
<Assembly: AssemblyDescription("Displays info from the AssemblyInfo.vb")>
<Assembly: AssemblyCompany("R 'n R - For Reading and Refreshment")>
<Assembly: AssemblyProduct("Books and Drinks")>
<Assembly: AssemblyCopyright("Copyright © 2003, All Rights Reserved")>
```

You can also change the Trademark information if you wish.

On the form add labels for Full Name, Title, Description, Company, Product, and Copyright. Write the code to assign values to the labels in the Form_Load event procedure.

```
'Project: Attributes
'Programmer: Bradley/Millspaugh
'Date: January 2003
'Description: Retrieve and display assembly attribute data

Option Strict On
Imports System.Reflection

Public Class frmMain
    Inherits System.Windows.Forms.Form
```
Private Sub frmMain_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
' Display attribute information
Dim objAttributes() As Object
Dim asmInfo As [Assembly]
Dim atrTitle As AssemblyTitleAttribute
Dim atrCompany As AssemblyCompanyAttribute
Dim atrProduct As AssemblyProductAttribute
Dim atrCopyright As AssemblyCopyrightAttribute
asmInfo = [Assembly].Load("AttributeDemo")
lblFullName.Text = "Assembly FullName:" & asmInfo.FullName
objAttributes = asmInfo.GetCustomAttributes(False)
Dim objItem As Object
' Convert from object to desired types
' Need Select Case because order of attributes in array varies
For Each objItem In objAttributes
    Select Case objItem.GetType.ToString()
        Case "System.Reflection.AssemblyTitleAttribute"
            atrTitle = CType(objItem, AssemblyTitleAttribute)
            lblTitle.Text = atrTitle.Title.ToString()
        Case "System.Reflection.AssemblyCompanyAttribute"
            atrCompany = CType(objItem, AssemblyCompanyAttribute)
            lblCompany.Text = atrCompany.Company.ToString()
        Case "System.Reflection.AssemblyProductAttribute"
            atrProduct = CType(objItem, AssemblyProductAttribute)
            lblProduct.Text = "We Sell:" & atrProduct.Product.ToString()
        Case "System.Reflection.AssemblyCopyrightAttribute"
            atrCopyright = CType(objItem, AssemblyCopyrightAttribute)
            lblCopyright.Text = atrCopyright.Copyright.ToString()
    End Select
Next
End Sub

Web Assembly Files

Web projects also contain the AssemblyInfo.vb file. You can use the same techniques to display the attribute information on a Web Form.

' Project: WebAttributes
' Programmer: Bradley/Millspaugh
' Date: January 2003
' Description Retrieve and display assembly attribute data
Option Strict On
Imports System.Reflection

Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents lblTrademark As System.Web.UI.WebControls.Label

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' Access assembly information
        Dim objAttributes() As Object
        Dim asmInfo As [Assembly]
Dim atrTrademark As AssemblyTrademarkAttribute
Dim objItem As Object

asmInfo = [Assembly].Load("WebAttributes")
objAttributes = asmInfo.GetCustomAttributes(True)
'Convert from object to the desired type
'Need Select Case because order in array varies
For Each objItem In objAttributes
    Select Case objItem.GetType.ToString()
        Case "System.Reflection.AssemblyTrademarkAttribute"
            atrTrademark = CType(objItem, AssemblyTrademarkAttribute)
            lblTrademark.Text = atrTrademark.Trademark.ToString()
    End Select
Next
End Sub

**Deploying Applications**

Most of this text is devoted to writing and testing applications using the VS IDE. However, once you get an application tested and ready for use, you will want to deploy it and run it on another computer. To run a .NET Windows application, the target computer must have the .NET Framework or the .NET Framework Redistributable installed. The Redistributable file is available on the Components CD of VS.NET in the DotNetFramework folder as dotnetfx.exe, and a free download on Microsoft's Web site. It is available for Windows 98, ME, NT, 2000, and XP. Notice that you can run .NET applications on Windows 98 and ME, even though the VS .NET IDE does not run on those operating systems.

You can choose from two methods for deploying your applications: (1) XCopy deployment or (2) Windows Installer technology. Microsoft Windows Installer is a separate application that ships with Windows and creates .msi files.

XCopy deployment gets its name from the old DOS XCOPY command, which copied all files in the current folder and all subfolders. Although you can use the XCOPY command for copying files, XCopy deployment in .NET simply means that you copy the necessary files from the development machine to the target machine.

Deploying a compiled Windows application can be as easy as copying the .exe file from the bin folder to another computer. However, deploying a Web application is a little more complicated because more than one file is needed to run an application.

To copy a Web application for deployment, you can manually copy the files you need or use the *Project / Copy Project* command (which appears on the menu only for Web projects). The files that you need to copy are at least *.aspx, global.asax, web.config, *.dll from the bin folder, and any images used by the application. If you use the *Project / Copy Project* command, all necessary files are copied to the destination that you choose. If you copy the project to a new folder in Inewpub/wwwroot on the host machine, the virtual directory is created for you. But if you copy to another location or move the project to another machine, you must set up the virtual directory in IIS. See Appendix E for help with setting up a virtual directory.
You can run a copied (deployed) project by typing its URL into the browser. For example, if you use Project / Copy Project to copy a Web project to a folder named RnrCustomers and a startup page of Main.aspx to Inetpub/wwwroot, type this URL into the browser:

http://localhost/RnrCustomers/Main.aspx

**Feedback 1.2**

Write the statements necessary to retrieve and display the copyright attribute in a label called lblCopyright.

**Helpful Hints for .NET**

Some of the new features of .NET can be handy for enhancing your projects. You have control over the form such as setting a default font and specifying the minimum and maximum sizes of the form.

**Default Font**

You can set a default font for your form, which applies to all controls on the form that do not have a font specifically assigned. You can set the form's Font property at design time or run time, either before or after adding controls to the form. As long as you don't set the Font property for an individual control, it will use the form's font. To change the form's default font in code, you can include a line similar to this one, generally in the Form_Load event procedure.

```vbnet
Me.Font = New Font("Arial", 14, FontStyle.Underline)
```

**Form Size**

.NET allows you to set a minimum and/or maximum size for your form. Your user can resize only within the limits that you allow. You can set the MinimumSize and MaximumSize properties at design time or in code.

```vbnet
Me.MinimumSize = New Size(200,200)
Me.MaximumSize = New Size(500,500)
```

**VB6 Compatibility Library**

When Microsoft upgraded VB from version 6 to .NET, they removed many features, so that VB code would be consistent for all CLR languages. Then they gave you a way to access many of the obsolete VB 6 features: the VB6 compatibility library. If you try to open a VB 6 application in VS .NET, the Upgrade
Wizard opens and attempts to convert your program from VB 6 to VB .NET. After the conversion you can look through your code and see that the wizard incorporated many features of the VB6 compatibility library and likely inserted some comments about statements that couldn’t be converted. The code looks bad, is inefficient, and may not run in future versions of VB. This statement comes directly from the MSDN Help files that accompany Visual Studio:

**Caution** Functions in the Visual Basic 6.0 Compatibility library are provided only for use by the upgrading tools. Although it is possible to use this library when writing new code, there is no guarantee that it will be supported in future versions of Visual Basic.

So, although you can use the old functions, it is not a good idea to do so. One example is the MsgBox function from VB 6. Although the Upgrade Wizard will keep the MsgBox statements, you should always use the new MessageBox object.

If you want to make sure that you are not using any features of the VB6 compatibility library, check the References node of the Solution Explorer and the Imports node in the Project Properties dialog box. If you find Microsoft.VisualBasic.Compatibility listed, delete it and see what error messages are generated.

Many professionals agree that the Upgrade Wizard causes more problems than it solves and should be avoided.

### Review Topics

This section is intended as a review for the chapter hands-on exercise that follows. You will create an MDI (multiple document interface) project with parent and child forms, a toolbar, and a status bar.

**MDI**

This short section is intended as a quick review of MDI in VB .NET. For a more in-depth review of MDI parent forms and child forms, see Appendix B.

- You can make any form into a parent form by setting its IsMdiContainer property to True. You can do this at design time.
- Make a form into a child form by setting its MdiParent property to the parent form. You must do this in code.

```vbnet
Dim frmChildOne As New frmChildOne()
frmChildOne.MdiParent = Me
frmChildOne.Show()
```

- A project can have multiple parent forms, multiple child forms, and forms that are independent. You may want to consider independent forms for such tasks as a splash form.
- When you close a parent form, all of its children also close.
- A child cannot wander outside of its parent’s area.
The Window Menu

A parent form should have a Window menu to list the open child windows and to allow the user to arrange multiple child windows. For an example, look at the Window menu in Word; you will see a list of the open documents as well as options for arranging the windows.

To make a menu show a list of the open child windows, set the MdiList property to True for the mnuWindow object.

Example

```
Window (mnuWindow)
Tile Horizontal (mnuWindowHorizontal)
Tile Vertical (mnuWindowVertical)
Cascade (mnuWindowCascade)
```

Layout Options

When several child windows are open, they may be arranged in several different layouts: tiled vertically, tiled horizontally, or cascaded. The type of alignment is set as an argument of the LayoutMdi method.

```
Me.LayoutMdi(MdiLayout.TileVertical)
Me.LayoutMdi(MdiLayout.TileHorizontal)
Me.LayoutMdi(MdiLayout.Cascade)
```

Redisplaying Child Windows

In some applications, such as a word processor, you may allow the user to open multiple child windows of the same type. In other applications, you want to allow only one child form of a particular type. For example, if your application has a Summary window that the user can display by selecting a menu option, he or she may select the option multiple times. The first time, you must instantiate the form, but the next time you should just activate the existing form, which brings it to the top of any other open child forms.

You can check for the existence of a child form by using the parent form’s MdiChildren property, which is a collection of all open child forms that belong to the parent.

```
' Determine if form already exists
Dim frmTest As Form
Dim blnFound As Boolean = False

' Does form already exist?
For Each frmTest In Me.MdiChildren
    With frmTest
        If .Name = "frmSummary" Then
            .Activate() ' Activate previous instance
            blnFound = True
        End If
    End With
Next
If Not blnFound Then
    Dim frmSummaryInstance As New frmSummary()
    With frmSummaryInstance
        .MdiParent = Me
        .Show()
    End With
End if
```
Image Lists

Use an ImageList component to store the images for toolbars and other controls. The component appears in the component tray and should have a prefix of “ils.”

To add images to the image list, select the Images collection Build button from the Properties window. Use the Add button in the Image Collection Editor to add each image to the collection. An index is automatically assigned to each image in the collection. When you assign the images to your toolbar buttons, you will see the index and a preview of each image.

Toolbars

Use the Toolbar icon in the toolbox to create a Toolbar object for your project. The toolbar itself does not have any buttons. You add those with a Buttons collection in the Properties window. Name your Toolbar object with “tlb” for the prefix; assign your image list component to the ImageList property of the toolbar.

To add buttons to the toolbar, simply click on the Build button for the Buttons collection and select the Add button from the ToolBarButton Collection Editor. When you drop down the list for the ImageIndex property, the index and images from the image list appear.

Coding a Toolbar

A Toolbar component has one ButtonClick event that occurs when the user clicks on any of the buttons. Use the procedure’s event arguments to determine which button was clicked. Assuming that the toolbar is named tlbParent, you can find the index of the selected button with tlbParent.Buttons.IndexOf(e.Button)

You can use the index of the selected button in a Select Case statement to send the user to the appropriate procedure.

    'Execute appropriate procedure for the button clicked
    Select Case tlbParent.Buttons.IndexOf(e.Button)
        Case 0
            mnuDisplayChildOne_Click(sender, e)
        Case 1
            mnuDisplayChildTwo_Click(sender, e)
    End Select
End Sub

Status Bars

A status bar is usually located at the bottom of a form to display information such as date, time, or status of the Cap Lock or Num Lock key. If you want a
status bar on your form, you need to take two steps: add a StatusBar control to your form and add StatusBarPanel objects to the status bar. Add a StatusBar control from the toolbox and name it using “sbr” as the prefix. By default, the ShowPanels property is set to False; change this to True.

Add StatusBarPanels to display the desired information by using the Panels collection. You may find that you need to adjust the panel width to display the information completely.

Assign the value that you want to display to the Text property of the panel. You can either name the panels or refer to them by their index.

```csharp
sbrParent.Panels(0).Text = Now.ToShortDateString()
```

Consider using the methods of the Now object for the date.

**Context Menus**

It’s easy to add context menus (also called shortcut menus or popup menus) to a form or controls. Use the regular menu designer to create a menu; the top-level menu should have a name but a blank Text property. Assign the menu name to the ContextMenu property of the form and/or any controls on the form. The easiest way to code the context menu is to add its event to the Handles clause of the similar selection from the main menu.

**Your Hands-On Programming Example**

Write an MDI project for R ’n R—For Reading and Refreshment. The project should have four forms: the Main form, the About form, the Payroll form, and the Summary form. The Payroll and Summary forms should have only a Close button. You will write code for the Payroll and Summary forms in Chapter 2.

The About form should display the company name and the copyright information from the assembly attributes.

**Menu**

File        View        Help
Exit        Payroll Form   About
Summary

Include a toolbar with buttons to display each of the forms: Payroll, Summary, and About. Each button should display an appropriate ToolTip. Also allow the user to display any of the forms from a context menu.

Display the current date in the status bar.
Planning the Project
Sketch the four forms for the application (Figure 1.4). Your users must sign off
the sketches as meeting their needs before you begin programming.

**Figure 1.4**
Sketch the forms for the R ’n R Payroll project: a. Main form (parent); b. Payroll form; c. Summary form; and d. About form.

Plan the Objects, Properties, and Methods
Determine the objects and property settings for the forms and controls. Figure
1.5 shows the diagram of the program classes.


**Figure 1.5**
The class diagram for the hands-on programming example.

<table>
<thead>
<tr>
<th>frmMain</th>
<th>frmPayroll</th>
<th>frmSummary</th>
<th>frmAbout</th>
</tr>
</thead>
<tbody>
<tr>
<td>mnuFileExit</td>
<td>btnClose</td>
<td>btnClose</td>
<td>lblCompany</td>
</tr>
<tr>
<td>mnuViewPayroll</td>
<td></td>
<td></td>
<td>lblCopyright</td>
</tr>
<tr>
<td>mnuViewSummary</td>
<td></td>
<td></td>
<td>PictureBox1</td>
</tr>
<tr>
<td>mnuHelpAbout</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tlbMain</td>
<td></td>
<td></td>
<td>frmAbout_Load</td>
</tr>
<tr>
<td>ilsRnR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sbrMain</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>frmMain_Load</th>
<th>mnuFileExit_Click</th>
<th>mnuHelpAbout_Click</th>
<th>mnuViewPayroll_Click</th>
<th>mnuViewSummary_Click</th>
<th>tlbMain_ButtonClick</th>
</tr>
</thead>
<tbody>
<tr>
<td>mnuFileExit</td>
<td>Close the form.</td>
<td>Create an instance of the About form.</td>
<td>Create an instance of the Payroll form.</td>
<td>Create an instance of the Summary form.</td>
<td>Execute the corresponding menu item event procedure.</td>
</tr>
<tr>
<td>mnuHelpAbout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mnuViewPayroll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mnuViewSummary</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tlbMain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**frmMain**

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmMain</td>
<td>Text</td>
<td>R’n R For Reading and Refreshment</td>
</tr>
<tr>
<td>frmMain</td>
<td>IsMdiContainer</td>
<td>True</td>
</tr>
<tr>
<td>mnuFileExit</td>
<td>Text</td>
<td>Exit</td>
</tr>
<tr>
<td>mnuViewPayroll</td>
<td>Text</td>
<td>&amp;Payroll Form</td>
</tr>
<tr>
<td>mnuViewSummary</td>
<td>Text</td>
<td>&amp;Summary</td>
</tr>
<tr>
<td>mnuHelpAbout</td>
<td>Text</td>
<td>&amp;About</td>
</tr>
<tr>
<td>tlbMain</td>
<td>Buttons collection</td>
<td>View Payroll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>View Summary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>View About</td>
</tr>
<tr>
<td>ilsRnR</td>
<td>ImageList</td>
<td>ilsRnR</td>
</tr>
<tr>
<td>sbrMain</td>
<td>Images collection</td>
<td>Images for 3 buttons</td>
</tr>
<tr>
<td>sbrMain</td>
<td>Panels collection</td>
<td>Add 1 panel for the date</td>
</tr>
</tbody>
</table>

**Procedure**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Actions—Pseudocode</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmMain_Load</td>
<td>Retrieve the date for the status bar.</td>
</tr>
<tr>
<td>mnuFileExit_Click</td>
<td>Close the form.</td>
</tr>
<tr>
<td>mnuHelpAbout_Click</td>
<td>Create an instance of the About form. Set the MdiParent property. Show the form.</td>
</tr>
<tr>
<td>mnuViewPayroll_Click</td>
<td>Create an instance of the Payroll form. Set the MdiParent property. Show the form.</td>
</tr>
<tr>
<td>mnuViewSummary_Click</td>
<td>Create an instance of the Summary form. Set the MdiParent property. Show the form.</td>
</tr>
<tr>
<td>tlbMain_ButtonClick</td>
<td>Execute the corresponding menu item event procedure.</td>
</tr>
</tbody>
</table>
### frmPayroll

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmPayroll</td>
<td>Text</td>
<td>Payroll</td>
</tr>
<tr>
<td></td>
<td>WindowState</td>
<td>Maximized</td>
</tr>
<tr>
<td>btnClose</td>
<td>Text</td>
<td>&amp;Close</td>
</tr>
</tbody>
</table>

#### Procedure Actions—Pseudocode

**btnClose_Click** Close the form.

### frmSummary

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmSummary</td>
<td>Text</td>
<td>Payroll Summary</td>
</tr>
<tr>
<td></td>
<td>WindowState</td>
<td>Maximized</td>
</tr>
<tr>
<td>btnClose</td>
<td>Text</td>
<td>&amp;Close</td>
</tr>
</tbody>
</table>

#### Procedure Actions—Pseudocode

**btnClose_Click** Close the form.

### frmAbout

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmAbout</td>
<td>FormBorderStyle</td>
<td>FixedDialog</td>
</tr>
<tr>
<td></td>
<td>StartPosition</td>
<td>CenterParent</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>About This Application (Changes at run time.)</td>
</tr>
<tr>
<td>lblCompany</td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td>lblCopyright</td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td>Label1</td>
<td>Size</td>
<td>290,1 (one-pixel-wide line)</td>
</tr>
<tr>
<td>PictureBox1</td>
<td>Image</td>
<td>StudentData/Graphics/Books.gif</td>
</tr>
<tr>
<td></td>
<td>SizeMode</td>
<td>StretchImage</td>
</tr>
</tbody>
</table>

#### Procedure Actions—Pseudocode

**frmAbout_Load** Retrieve the attributes and set up the labels.

### Write the Project

Following the sketches in Figure 1.4, create the forms. Figure 1.6 shows the completed forms.

- Set the properties of each of the objects, as you have planned.
- Write the code for the forms. Working from the pseudocode, write each procedure.
- Modify the AssemblyInfo.vb file to hold the company attributes.
- When you complete the code, test each of the options. Make sure that all menu items work, the context menus work, and ToolTips appear for each button.
The forms for the R ’n R Payroll project: a. Main form (parent); b. Payroll form; c. Summary form; and d. About form.

The Project Coding Solution

frmMain

'Project: Ch01HandsOn
'Form: frmMain
'Programmer: Bradley/Millsapugh
'Date: January 2003
'Description: MDI Parent form; contains the menu and displays the various forms.

Option Strict On

Public Class frmMain
    Inherits System.Windows.Forms.Form
Private Sub mnuFileExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles mnuFileExit.Click
    'Close all forms
    Me.Close()
End Sub

Private Sub mnuHelpAbout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles mnuHelpAbout.Click, mnuAbout.Click
    'Display the About box
    Dim blnFormExists As Boolean = False
    blnFormExists = CheckFormExistence("frmAbout")
    If Not blnFormExists Then
        Dim frmAboutInstance As New frmAbout()
        With frmAboutInstance
            .MdiParent = Me
            .Show()
        End With
    End If
End Sub

Private Sub mnuViewPayroll_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles mnuViewPayroll.Click, mnuPayroll.Click
    'Display the Payroll form
    Dim blnFormExists As Boolean = False
    blnFormExists = CheckFormExistence("frmPayroll")
    If Not blnFormExists Then
        Dim frmPayrollInstance As New frmPayroll()
        With frmPayrollInstance
            .MdiParent = Me
            .Show()
        End With
    End If
End Sub

Private Sub mnuViewSummary_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles mnuViewSummary.Click, mnuSummary.Click
    'Display the Summary form
    Dim blnFormExists As Boolean = False
    blnFormExists = CheckFormExistence("frmSummary")
    If Not blnFormExists Then
        Dim frmSummaryInstance As New frmSummary()
        With frmSummaryInstance
            .MdiParent = Me
            .Show()
        End With
    End If
End Sub

Private Sub tlbMain_ButtonClick(ByVal sender As System.Object, ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) Handles tlbMain.ButtonClick
    'Execute the appropriate event procedure
    Select Case tlbMain.Buttons.IndexOf(e.Button)
        Case 0
            mnuViewPayroll_Click(sender, e)
Case 1
mnuViewSummary_Click(sender, e)
Case 2
mnuHelpAbout_Click(sender, e)
End Select
End Sub

Private Sub frmMain_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
'Get the date for the status bar
sbrMain.Panels(0).Text = Now.ToShortDateString
End Sub

Private Function CheckFormExistence(ByVal strFormName As String) As Boolean
' Determine if form already exists
Dim frmTest As Form
Dim blnFound As Boolean = False

'Does form already exist?
For Each frmTest In Me.MdiChildren
With frmTest
If .Name = strFormName Then
 Activate()
 blnFound = True
End If
End With
Next
Return blnFound
End Function
End Class

frmPayroll

'Project: Ch01HandsOn
'Module Payroll Form
'Programmer Bradley/Millspaugh
'Date January 2003
'Description: User interface for the payroll information.

Option Strict On
Public Class frmPayroll
Inherits System.Windows.Forms.Form

Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnClose.Click
' Close this form and return to the parent form
Me.Close()
End Sub
End Class

frmSummary

'Project: Ch01HandsOn
'Module Summary Form
'Programmer Bradley/Millspaugh
'Date January 2003
'Description: User interface to display the payroll summary information.
Public Class frmSummary
    Inherits System.Windows.Forms.Form

    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnClose.Click
        'Close this form
        Me.Close()
    End Sub
End Class

frmAbout

'Project: Ch01HandsOn
'Form: frmAbout
'Programmer: Bradley/Millsop
'Date: January 2003
'Description: Display information about the application from the attributes in the AssemblyInfo.vb file.

Option Strict On

Imports System.Reflection

Public Class frmAbout
    Inherits System.Windows.Forms.Form

    Private Sub frmAbout_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'Load the labels from AssemblyInfo.vb
        'Display attribute information
        Dim objAttributes() As Object
        Dim asmInfo As [Assembly]
        Dim atrCompany As AssemblyCompanyAttribute
        Dim atrCopyright As AssemblyCopyrightAttribute
        Dim atrTitle As AssemblyTitleAttribute
        asmInfo = [Assembly].Load("Ch01HandsOn")
        objAttributes = asmInfo.GetCustomAttributes(False)
        Dim objItem As Object
        'Convert from object to desired types
        'Need Select Case because order of attributes in array varies
        For Each objItem In objAttributes
            Select Case objItem.GetType.ToString()
                Case "System.Reflection.AssemblyTitleAttribute"
                    atrTitle = CType(objItem, AssemblyTitleAttribute)
                    Me.Text = "About" & atrTitle.Title.ToString()
                Case "System.Reflection.AssemblyCompanyAttribute"
                    atrCompany = CType(objItem, AssemblyCompanyAttribute)
                    lblCompany.Text = atrCompany.Company.ToString()
                Case "System.Reflection.AssemblyCopyrightAttribute"
                    atrCopyright = CType(objItem, AssemblyCopyrightAttribute)
                    lblCopyright.Text = atrCopyright.Copyright.ToString()
            End Select
        Next
    End Sub
End Class
Summary

1. The .NET Framework SDK contains the class libraries, the common language runtime, and ASP.NET.
2. Managed code is compiled to run in the common language runtime.
3. A portable executable file contains intermediate language (managed code) and metadata.
4. Metadata store information about the methods, classes, and types for the run-time PE file.
5. The .NET Framework is composed of a set of classes stored in the class library. The classes are organized into a hierarchy of namespaces.
6. The CLR treats value types and reference types differently. Each value type variable has the value stored in the variable’s memory location and is always initialized. A reference type variable holds a pointer to an actual object and may be equal to Nothing if not assigned.
7. The compiler produces MSIL (Microsoft Intermediate Language), a platform-independent set of instructions.
8. An assembly is the smallest deployable unit of code, which contains one or more .exe or .dll files and a manifest that describes the assembly.
9. The AssemblyInfo.vb file holds attributes, which are tags that contain information about the assembly. Both Windows applications and Web pages can access the attribute information at run time.
10. A Reference object connects Visual Basic to external components, either assemblies or COM objects.
11. Web development is done using ASP.NET.
12. A compiled program becomes an .exe or .dll file in the bin folder of the project. The .resx file holds resources for the form, including any graphics.
13. You can retrieve and display the attributes from AssemblyInfo.vb at run time in a Windows or a Web application.
14. You can deploy a Windows application by copying the .exe file to another location. For a Web application, more files are needed. Use the Copy Project menu item to copy all needed files.
15. Setting the Font property of a form sets the default font for all controls on the form that do not have a specific font assigned.
16. Forms can have a minimum size and/or a maximum size.
17. It is not advised to use the VB6 compatibility library.
Review Questions

1. Differentiate between the Framework and Visual Studio.
2. Explain the following:
   a. CRL
   b. CLS
   c. PE
   d. MSIL
   e. ASP.NET
3. Explain the relationship between the common language runtime and managed code.
4. What is the purpose of compiling to an intermediate language?
5. What is the difference between a value type and a reference type?
6. What is an assembly? What does an assembly contain?
7. What are attributes? Give three examples.
8. What is ASP.NET and what is its purpose?
9. Where can you find a project’s compiled version? What else might you find in that same location?
10. Explain how to display the attributes in AssemblyInfo.vb on a form at runtime.
11. How can you deploy Windows applications? Web applications?
12. How can you set the default font for all controls on a form?
13. Why should you avoid the functions in the VB6 compatibility library?
14. What is an MDI application? How many parent forms can be in a single MDI application?

Programming Exercises

1.1 Create a Web project that displays the company, title, and copyright information from the attributes on the page. Make up your own information and modify the attributes file.

1.2 Create a Windows application that displays the company, title, and copyright information from the attributes on the form. Make up your own information and modify the attributes file.
1.3 Create a multiple form project for Tricia's Travels that contains a Main form and an About form. The main form will have a menu that contains the following:

Menu

File    Edit    Window    Help
Exit    Guests    Tile Horizontal    About
Rooms    Tile Vertical
Charges    Cascade

Change the default font of the main form to a font of your choice. Make the About form show the company and copyright information from the attributes. Make up your own values and modify the attribute file.

Case Studies

Claytor's Cottages

Create a project for Claytor's Cottages, a small bed and breakfast. Use an MDI form with a menu, a toolbar, and a status bar.

The About form should be a child form and contain at least your name, copyright, and company name, taken from the attributes.

Create child forms for each option (Guests, Rooms, Reservations) that simply have the title bar caption indicating the form purpose and a Close button to return to the main form. \textit{Note:} These forms will be modified in later chapters.

Menu

File    Edit    Window    Help
Exit    Guests    Tile Horizontal    About
Rooms    Tile Vertical
Charges    Cascade

Select the window list option that allows the open forms to display on the \textit{Window} menu.

Toolbar

Include three buttons to open each of the child forms. Place ToolTips on each button.

Guests
Rooms
Charges

Status Bar

Include the date and the time of day at the right side of the toolbar. Leave a panel for text messages to the left side.

Context Menu

Create a context menu on the parent form that has options to display the Guests, Rooms, and Charges forms.

Standards

- Use standard prefixes for naming variables, objects, and procedures.
- Menu items and controls must have keyboard access. Use standard selections when appropriate.
- Use a maximized window state for the main form.
- Set the form's Icon property to an appropriate icon.
Create a project for Christian's Car Rentals. The project should contain an MDI Main form with a menu, a toolbar, and a status bar.

The About form should be a child form and contain at least your name, copyright, and company name, taken from the attributes.

Create child forms for each option (Customers, Vehicles, and Rentals) that simply have the title bar caption indicating the form purpose and a Close button to return to the main form. Note: These forms will be modified in later chapters.

Menu

File Edit Window Help
Exit Customers Tile Horizontal About
Vehicles Tile Vertical
Rentals Cascade

Include keyboard shortcuts for all menu options, following standards where applicable. Select the window list option that allows the open forms to display on the Window menu.

Toolbar

Place three buttons on the toolbar, one to display each of the child forms. Use any icon that you wish for each of the buttons and include ToolTips for each.

Customers
Vehicles
Rentals

Status Bar

Include the date in a panel at the right end of the toolbar. Leave a panel for text messages at the left end.

Context Menu

Create a context menu on the parent form that has options to display the Customers, Vehicles, and Rentals forms.

Standards

• Use standard prefixes for naming variables, objects, and procedures.
• Menu items and controls must have keyboard access. Use standard selections when appropriate.
• Use a maximized window state for the main form.
• Set the form's Icon property to an appropriate icon.