August 26, 2009

Section II: Introduction to VB.NET

Algorithm Development

"The ability to learn faster than your competitors may be the only sustainable competitive advantage."

- *Peter Senge*

Chapter 3: Getting Started with .NET

In this chapter, you will learn to maneuver around the .NET integrated development environment (IDE) and how to customize it to your liking for efficient development. While we will only be writing a smidgen of code, we will be creating a professional looking program and learning a few simple techniques that are big time savers.

3.1 The .NET Integrated Development Environment

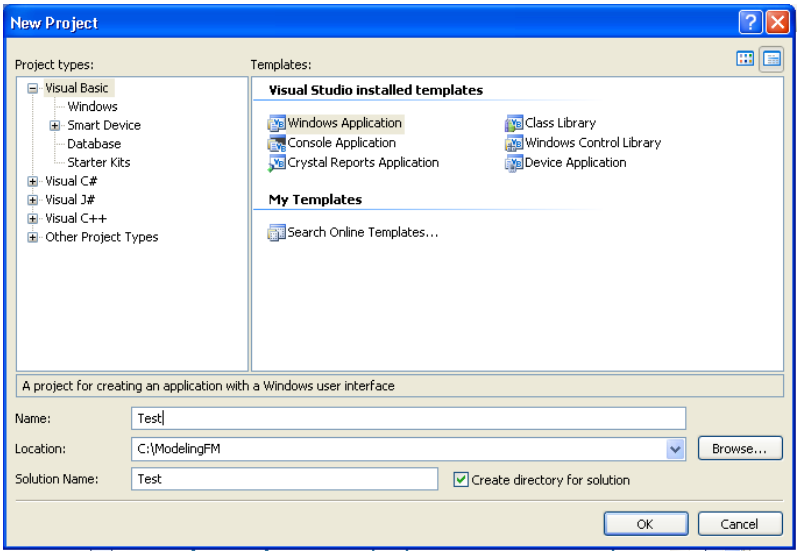
Visual Studio.NET enables you to program visually, dragging and dropping controls, like buttons and text boxes, into place rather than creating them in code. In this way, visual programming greatly increases programmer productivity. Visual Studio.NET also includes several advanced tools for writing and debugging your program code. Let's jump right in.

Step 1 Before you open a .NET solution, you will need to create a separate folder on your hard drive to hold all the files for all the projects in this book. Create a folder called "C:\ModelingFM"

Step 2 Now, go ahead and open Visual Studio.NET.

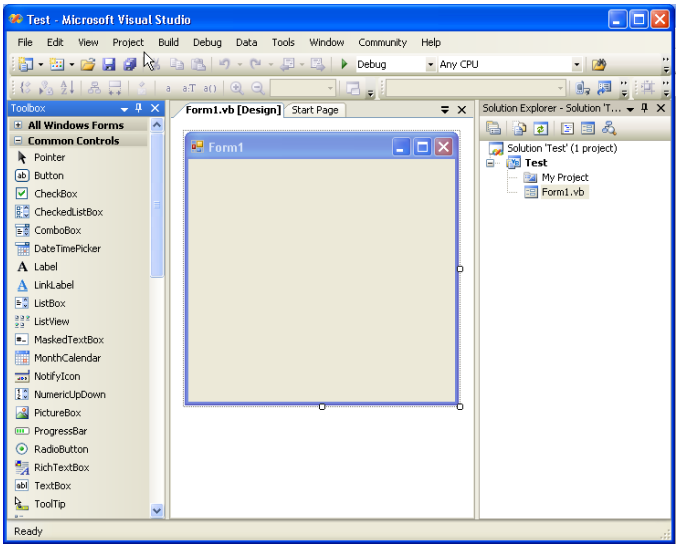
Step 3 When the Start Page opens, on the menu bar, select File, New, and Project…

Step 4 In the New Project window, select Visual Basic (or Visual C#) as the Project type and Windows Application as the template. Give the project the name "Test" and the location of the ModelingFM folder. Later in the book we will look at some of the other templates.



When a new project is created, .NET automatically places all the files associated with your new project within a folder of the same name. So, the path to your new project should be C:\ModelingFM\Test\. If you take a look at the contents of this folder via Windows Explorer, you will notice that several files and subfolders have been created to contain all the elements of our project. .NET applications that we build consist of several files. We will learn more about some of these files in later chapters. For right now, just be aware that programs consist of several files in a folder. To later reopen the project for further development, click on the file with the .sln extension.

Let's take a look at the .NET IDE that should now be visible on your screen. Notice that the development environment consists of several windows, which are all either dockable or free-floating, allowing you to customize the environment to your liking. The form in the center, labeled Form1, is where we will actually build the graphical user interface for our program.



3.2 Menu Bar

Across the top the screen is the menu bar. Take some time to peruse the menu bar and become familiar with the types of commands that perform various actions. Many of these commands also have corresponding short cuts, either through key strokes or menu bar icons or both. As you will no doubt discover as you gain experience in programming in the .NET IDE, there are often several ways to accomplish the same task.

3.3 Toolbox

Written vertically down the left side of the screen should be the "Toolbox" button. If you do not see it, click on the toolbox icon in the upper right hand corner. It's the one with a hammer and wrench in an X shaped design. When you open the toolbox, you will see the lists of tools, called controls, that you see in the picture above. We will often be adding controls, by "dragging and dropping" them into our forms, to rapidly build programs and graphical user interfaces (GUIs). You may want to spend a little time investigating each of the tools before you proceed.

3.4 Solution Explorer Window

The Solution Explorer window, shown in the upper right corner, enables you to access the different parts of your project. If the Solution Explorer window is not visible, click on the Solution Explorer icon or, on the menu bar, click View and Solution Explorer. In our applications, we almost always have several forms and classes and program modules. The Solution Explorer gives us instant access to any part of our project at any time. To close the Solution Explorer, click the "X" button in the upper right hand corner.

3.5 Properties Window

In the lower right corner is the Properties window. Again, if it is not visible, click on the Properties Window icon on the toolbar or select View and Properties Window from the menu bar.

Properties are attributes, like size and color, of the objects we use in programs. Since each control, or tool, from our toolbox is an object and has its own set of properties we can see all of the properties associated with each of them in this window. You should familiarize yourself with the different properties associated with the different controls as we use them throughout this book. The left hand of the Properties Window column lists the individual properties and the right hand column lists the value of each property. You will need the Properties Window to set the initial values of these properties at design time and, as you will later see, we can change properties at run time using .NET code. As with Solution Explorer, we can close the Properties window and reopen it either from the View menu or the menu bar icon.

Other, non-visible objects we create in our programs will also have properties associated with them. When we cannot see the objects, it gets slightly more difficult to understand properties. For example, in finance, a call option could be an object. An option object in our program would certainly have properties, like an option symbol, strike price, expiration date, and implied volatility.

3.6 Events

Besides properties, controls also have events associated with them. An event is triggered when something happens to a control. The button "click" event is the probably the most easily understandable example of an event. Later, we will learn how to program things to happen when events are fired.

3.8 Methods

Objects, like controls, also have to perform functions of their own. It isn't usually enough that an object simply exists. After all, the whole point of creating a control is that the object does something useful. These additional functions are known as methods in .NET terms. Whereas properties are thought of as nouns, methods are often thought of as verbs. Later, we will learn how to create our own objects and add methods to them.

3.9 Visual Studio.NET Help

There is no way that any book can hope to cover all the features of .NET or all the potential instances you may uncover for using them. Finding and solving new problems quickly is one of the joys of programming. Fortunately, Visual Studio.NET provides a vast array of help features. Knowing how to find what you need in the Help files is one of the most valuable skills you can gain to improve your expertise. Again, you should investigate the help files on your own and become comfortable accessing the Help index and Dynamic Help. Most often, programming questions that arise are covered extensively in the Help files, almost always with code examples.

3.10 Creating an Executable Program

You will write many different applications as you go through this book. Creating an executable program allows you to run your application as a single .exe file from the Windows environment without having to be in the .NET IDE. In order to create an executable file, .NET programs must be compiled into machine language.

Compiling a .NET application is a two-stage process. First, the program is compiled into Microsoft Intermediate Language (MSIL). Then second, another compiler translates this MSIL code into a single, executable file in machine language. In this way, Microsoft's .NET framework provides language interoperability. So, different parts of a program, written in different languages, can be combined to create a single program. In fact, any .NET-compliant language can be compiled into MSIL in this way and thus .NET is said to be language independent.

Step 5 Make sure your form, known by the default name "Form1" in your project, is active by clicking on it. You can change the size of the form by pulling on the highlighted corners or sides. This will automatically cause the Size property of the form to change. Now, in the Properties window, change the value of the "Text" property to read "My First .NET Program" without the quotation marks. When you press enter, you will see the title on the form change to the new text.

Step 6 In the Toolbox, click on Label and "paint" a label on your form by holding down the left mouse button and dragging over the form.

Step 7 This new label is now known by the default name, Label1, as you can see in the Properties window. Make sure the label is selected, and in the Properties window, change the Font property to Garamond, Bold, size 24. Also, change the Text property to "Buy Low, Sell High" Change the TextAlign property to MiddleCenter.

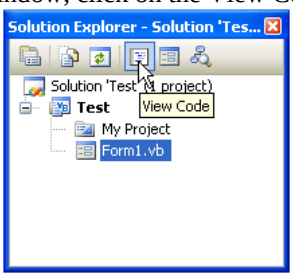
Your form should now look like this:

Step 8 Now to run your program, click the Start button on the menu bar, or under the Debug tab, click start. The Start button is the one that looks like a blue arrow, next to the word "Debug." Your program should take a few seconds to compile, and then it will run. You can close the program by clicking on the "X."

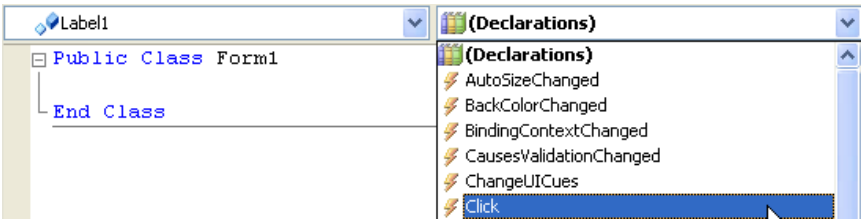
Step 9 In Windows Explorer you can find the executable program, Test.exe, in the Test folder, subfolder bin. The path to the file in full should be C:\ModelingFM\Test\bin\Test.exe. If you close down the Visual Studio IDE, you can run this executable program by double clicking it. Furthermore, you can drag the Test.exe icon onto your Windows desktop. You can even email it to your friends so that they never forget how to make money in the markets.

Now, let's take a little deeper look at the Visual Studio IDE.

Step 10 If you have not already done so, close the program, so that you are back in the Visual Studio IDE. In the Solution Explorer window, click on the View Code icon as shown.



Step 11 The Form1 code window will appear. This is where we write .NET code that is associated with the controls we place on Form1, including code that runs when events happen, as previously discussed.



In the combo boxes across the top of the code window, click on Label1 in the left hand combo box, and open the list in the right hand combo box. This is a list of all the events associated with our label, Label1. All the controls in the Toolbox have events associated with them. When an event happens, we can add code to make something happen.

Step 12 For example, select Click from the list of events for Label1. Notice that Visual Studio writes a stub of the event handler for you. In the event code routine, type Label1.Text = "Sell High, Buy Low." The underscores you see below allow us to wrap long lines of code onto the next line.

VB

```
Private Sub Label1_Click(ByVal sender As Object,
                         ByVal e As System.EventArgs)
                         Handles Label1.Click
     Label1.Text = "Sell High, Buy Low"
End Sub
```

C#

```
private void label1_Click( object sender, Event Args e )
{
     label1.Text = "Sell High, Buy Low";
```

```
}
```

Step 13 Run the program again. Notice that the program runs the same as previously. But, if you double click on the label in the form, an entirely new way to profit in the markets appears.

3.11 .NET Framework

What we really create in VB.NET code are types. The term "type" represents a broader description of any combination of data storage and functionality. Classes are but one example of a type, as are variables and functions. Instances of types allocate space for data storage and provide us with the behaviors we require. Deciding what types to use—classes, modules, subroutines, functions, structures, etc.--in our programs for data manipulation will be the focus of the remainder of the technology portions of the book

3.12 Types

A type is a generic term used to describe a representation of a value. Instances of types, in their various forms, encapsulate all the logic in our programs, so fully understanding types is fundamental to higher level .NET programming, not just VB.NET. Through the .NET Framework's common language specification and common type system, it is easy to use several different languages to create a single application, although this book is only concerned with Visual Basic.NET. This common type system looks like this:

.NET Common Type System	Example
Value Types	Variables, constants, structures, et al.
Reference Types	Objects

3.13 Interfaces

An interface specifies a group of methods that can only be implemented by another class or structure. We cannot then instantiate interfaces by themselves. As a practical matter, predefined .NET interfaces start with the letter I, as in ICollection, IList and IComparable. In the VB.NET help files you can survey the different interfaces and their respective members. In addition, we can declare our own, user-defined interfaces. Here is an example:

```
Interface ITradable
    Function Buy(ByVal Price as String) as Double
    Function Sell(ByVal Price as String) as Double
End Interface
```

The ITradable interface indicates that we can buy or sell something, but does not define how it happens. So, different financial instruments have the ability to be traded electronically, and, therefore, as objects should implement the ITradable interface. We have not though specified exactly how they will be traded, since the implementation of a trade for different instruments may be very different. For example, routing a buy order to the ISE may require a much different implementation than say routing a buy order to the CME. So, we have deferred the implementation of the interface to the definition of the class, which implements the interface. Nonetheless, the "stub" is there. We may at some point then implement an interface like this:

```
Class InstrObj
     Implements ITradable
…
Public Function Buy(ByVal Price as String) as Double_
      Implements ITradable.Buy
…
End Function

End Class
```

Classes may implement multiple interfaces although interfaces do not support multiple inheritance in derived classes. If a class definition includes that of ITradable, the buy and sell interfaces, and an object is instantiated, we can call the sell method:

```
Dim m_Instr as New InstrObj()
m_Instr.Sell("Market")
```

3.14 Assemblies

Assemblies are the fundamental building blocks of VB.NET applications and are held in executable files (EXEs) or dynamic link library files (DLLs). An assembly is a collection of types, which can be modules, interfaces, classes, delegates, enumerations, structures and other units of functionality. When we create a VB.NET application, the most common assemblies are already referenced for us. However, if we need to use an assembly that is not already referenced, we need to add a reference to the corresponding DLL file and use the Imports statement for the appropriate namespace. Once we have added a reference to the assembly and imported the namespace, all the classes, properties, methods, and other types in the namespaces are available to our application as if the assembly's code were part of it. Be aware that a single assembly might contain multiple namespaces, and each namespace might contain multiple types.

3.15 Namespaces

The VB.NET Framework class library is made up of namespaces, which contain and organize types, which are defined in an assembly. If two classes have the same name, we can still use them both as long as they are in different namespaces and we qualify the class names using the namespace. "Fully qualified" object names are prefixed with the name of the namespace where the object is defined. So, for example, System.Windows.Forms.ListBox is the fully qualified name of the listbox class since we have included the namespace. All namespaces in VB.NET begin with either System or Microsoft.

3.16 Imports Statement

The Imports statement does not itself provide access to assemblies, but rather simplifies access to them by eliminating the need to fully qualify named references. That is, we can use types defined within the imported namespace of the assembly without qualification. A module may contain any number of references and Imports statements, as long as the Imports statements appear after any Option statements and before any other code. For example,

```
Imports System.Data.OleDb
```

3.11 Summary

.NET makes every effort to provide us with the tools that simplify and speed the process of creating our own applications, or solutions as they are called. If you already program in a previous version of Visual Basic you will notice several similarities in the Visual Studio IDE. If you are new to programming, you will be able to turn out professional-looking applications even while you are learning VB.NET or C#.

Make sure you practice using the Help files. Practically everything you need to know is included in there somewhere. You might have to dig for it, but it is in there somewhere.

In the example program in this chapter we looked at the label control and the properties and events associated with it. We even wrote a brief statement to change the text property when the double click event is fired.

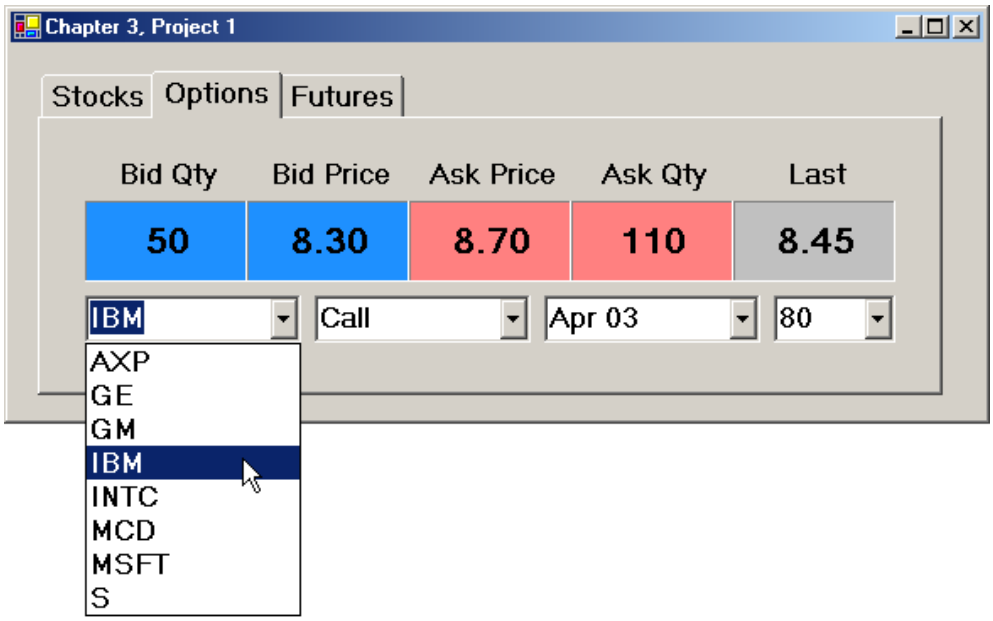
Chapter Three Problems

1. Where does .NET store the various files associated with your program?
2. Where will you find the controls used to create a graphical user interface?
3. If the Properties window is closed, how can you reopen it?
4. How do you create an executable program?

5. What are properties, events and methods?

Project One

Create a graphical user interface like the one pictured below. Use a tab control with three tab pages named Stocks, Options and Futures. On the Options tab page, place four combo boxes with sorted items, including 10 stock tickers, Put/Call, 12 expirations and 10 strikes. Also, place labels on your tab forms with Fixed3D border style and custom background colors. Name the colored labels lblBidQty, lblBid, lblAsk, lblAskQty, and lblLast. Try adding some controls and changing the fonts to enhance the appearance and user-friendliness of your GUI. Also, on the Stocks and Futures tab pages, add similar content for financial instruments of these types. For right now, to keep things simple do not add any code to handle any of the events associated with the controls on your form.



Project Two

Use buttons, group boxes and radio buttons to create the GUI pictured below. The default checked property for each of your On buttons should be set to true. When you run the program and check the False radio buttons, the true buttons should turn off automatically because they are grouped together. For right now, to keep things simple do not add any code to handle any of the events associated with the controls on your form.