CHAPTER 15

Programming with Windows Forms Controls

This chapter is concerned with providing a roadmap of the suite of GUI widgets defined in the System.Windows.Forms namespace. You have already had a chance to work with some Form-level control types such as MainMenu, MenuItem, StatusBar, and ToolBar (see Chapter 13); however, in this chapter, you will be examining various types that tend to exist within the boundaries of a Form’s client area (e.g., Buttons, TextBoxes, Panels, and the like).

In addition to giving you a formal grounding in the Windows Forms Control set, this chapter also details a number of related topics, such as establishing the tab order for your widgets, as well as configuring the docking and anchoring behaviors for your family of GUI types. The chapter then investigates the process of building custom dialog boxes, including techniques for responding to (and validating) user input.

Next, you’ll examine a new facility offered by the .NET Windows Forms architecture: Form inheritance. As you will see, it is now possible to establish “is-a” relationships between related Forms (which should make the MFC developers of the world insanely happy). Finally, we wrap things up with an examination of the process of building custom Windows Forms controls (which should make the ATL developers of the world insanely happy).

Understanding the Windows Forms Control Hierarchy

The System.Windows.Forms namespace contains a number of types that represent common GUI widgets that allow you to respond to user input in a Windows Forms application. Because .NET is a system of types built on standard OO principles, these controls are arranged in a hierarchy of related types. Figure 15-1 illustrates the big picture (note that System.Windows.Forms.Control is the common base class for all Windows Forms widgets.)
As you learned in Chapter 13, the System.Windows.Forms.Control type is the base class that provides a minimal and complete set of behaviors for all descending widgets. This includes the ability to process mouse and keyboard events, establish the physical dimensions of the widget using various properties (Height, Width, Left, Right, Location, and so on), manipulate background and foreground colors, establish the active font, and so forth. Also, the Control base type also defines members that control a widget’s anchoring and docking behaviors (as seen later in this chapter).

As you read through this chapter, remember that the widgets examined in this chapter gain a good deal of their functionality from the System.Windows.Forms.Control base class. In this chapter we’ll focus (more or less) on a given type’s unique members.

Adding Controls to Forms (IDE-Free)

Regardless of which type of control you choose to place on a Form, you will follow a similar set of steps. First of all, you must define member variables that represent the GUI widgets maintained by the Form. Next, inside the Form’s constructor (or within a helper method, called by the constructor), you’ll configure the look and feel of each control using the exposed properties, methods, and events. Finally (and most important), once the control has been set to its initial state, it must be added into the Form’s internal controls collection using the inherited Controls property. If you forget this final step, your widgets will not be visible at runtime! To illustrate this process in the raw, consider the MyForm class:

```csharp
// Don't forget to add a reference to System.Windows.Forms.dll!
using System.Windows.Forms;

class MyForm : Form
{
```

---

Figure 15-1. The Windows Forms control hierarchy