Ordering Information: **C# How to Program** & **The Complete C# Training Course**

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ON-LINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html** To learn more about Deitel instructor-led corporate training delivered at your location, visit **www.deitel.com/training** or contact Christi Kelsey at (978) 461-5880 or e-mail: **christi.kelsey@deitel.net.**

Note from the Authors: This article is an excerpt from Chapter 4, Section 4.13 of *C# How to Program*. This article introduces some basic concepts on manipulating properties in a Windows application. Readers should be familiar with the basics of C# syntax, as well as how to add controls to a Form in Visual Studio .NET, and customize those controls via the Properties window. The code examples included in this article show readers examples using the Deitel™ signature *LIVE-CODE™ Approach*, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

4.13 Introduction to Windows Application Programming

Today, users demand software with rich graphical user interfaces (GUIs) that allow them to click buttons, select items from menus and much more. Most C# programs used in industry are Windows applications with GUIs.

Load the project **`ASimpleProject`** into the IDE. This code can be downloaded, along with the rest of the code for *C# How to Program*, under the **Downloads/Resources** link off of our Web site (**`www.deitel.com`**). To identify easily the form and its controls in the program code, change the **`Name`** properties of the form, label and picture box to **`ASimpleProgram`**, **`welcomeLabel`** and **`bugPictureBox`**, respectively. [Note: This property may appear as **(Name)** in the **Properties** window.] To change a GUI component's properties, select (click) the component in the design window, then locate the property in the **Properties** window. Click the box to the right of the property name to input a new value, and press the *Enter* key once the new value has been entered.

With visual programming, the IDE generates the program code that creates the GUI. This code contains instructions for the creation of the form and every control on it. Unlike a console application, a Windows application's program code is not displayed initially in the editor window. Once the program's project (e.g., **`ASimpleProgram`**) is opened in the IDE, the program code can be viewed by selecting **View > Code**. Figure 4.16 shows the code editor displaying the program code.

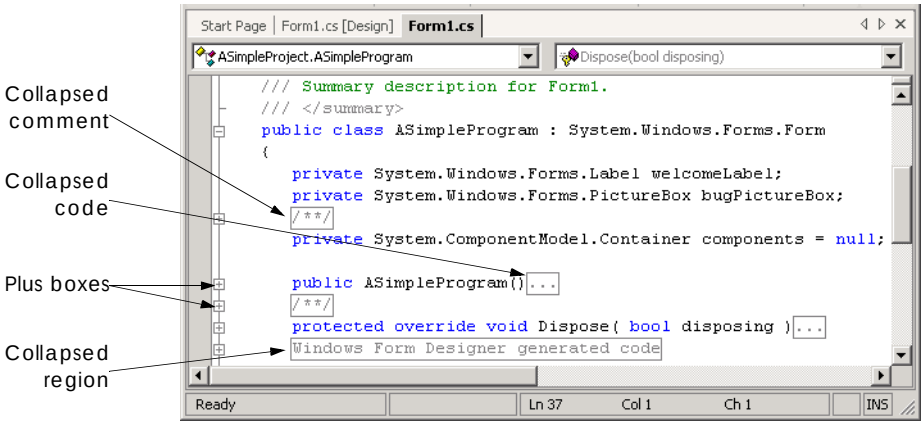

Fig. 4.16 IDE showing program code for **`ASimpleProgram`**.

Every Windows application consists of at least one class that *inherits* from class ***`Form`*** (which represents a form) in the .NET Framework Class Library's **`System.Windows.Forms`** namespace. The keyword **`class`** begins a class definition and is followed immediately by the class name (**`ASimpleProgram`**). Recall that the form's name is set using the **`Name`** property. A colon (**`:`**) indicates that the class **`ASimpleProgram`** inherits existing pieces from another class. The class from which **`ASimpleProgram`** inherits—here, **`System.Windows.Forms.Form`**—appears to the right of the colon. In this inher-

itance relationship, **Form** is called the *base class* (or *superclass*), and **ASimpleProgram** is called the *derived class* (or *subclass*). With inheritance **ASimpleProgram**'s class definition has the *attributes* (data) and *behaviors* (methods) of class **Form**. [*Note*: Changing a control's name in the **Properties** window may not change all occurrences of the control's name in the code. The reader should search the code and replace names that were not changed by the IDE. For example, the original form name (and class name) was **Form1**. Search the code for **Form1** and change any remaining instances to **ASimpleProgram**.]

A key benefit of inheriting from class **Form** is that someone else has previously defined "what it means to be a form." The Windows operating system expects every window (e.g., form) to have certain attributes and behaviors. However, because class **Form** already provides those capabilities, programmers do not need to "reinvent the wheel" by defining all those capabilities themselves. In fact, class **Form** has hundreds of methods! In our programs up to this point, we have used only one method (i.e., **Main**), so you can imagine how much work went into creating class **Form**. The use of the colon to extend from class **Form** enables programmers to create forms quickly.

In the editor window (Fig. 4.16), notice that portions of text are enclosed in small rectangles with small plus boxes to their left. The plus box indicates that this section of code is *collapsed*. Although collapsed code is not visible, it is still part of the program. Code collapsing allows programmers to hide code in the editor, so that they can focus on smaller (and perhaps more important) code segments. Clicking the plus box expands the code (i.e., displays the entire segment of code). A small minus box to the left of expanded code can be clicked to collapse that portion of code. If programmers wish to briefly view collapsed code, they can place the cursor over the rectangle for that portion of code. Doing so will display the code in a small window.

The appearance of collapsed code differs depending on the nature of the code. Figure 4.16 demonstrates that collapsed comments appear as **/**/** within a rectangle, while collapsed statements appear as an ellipses (**...**) within a rectangle. Finally, *regions* of code can be created in a C# program. A region is a portion of code specified between the text **#region** and **#endregion**. These tags specify the text that appears in the rectangle when the code is collapsed. One such region is shown at the bottom of Fig. 4.16, containing the text **Windows Form Designer generated code**. The description in the rectangle indicates that the collapsed code was created by the *Windows Form Designer* (i.e., the part of the IDE that creates the code for the GUI). This collapsed code contains the code created by the IDE for the form and its controls, as well as code that enables the program to run. We investigate this region momentarily.

Upon initial inspection, the *expanded code* (Fig. 4.17) looks complex. This code is created by the IDE and normally is not edited by the programmer. Such code is present in every Windows application. Allowing the IDE to create this code saves the programmer considerable development time. The reader is not at this point expected to understand how this code works. However, certain programming constructs, such as comments and control structures, should be familiar.

In Fig. 4.17 we see a portion of the code that makes up the region we saw in the previous figure. Notice that the region begins with the text **#region**, followed by the text that will be displayed when the region is collapsed. This region ends when the text **#endregion** is encountered, after the end of method **InitializeComponent**.

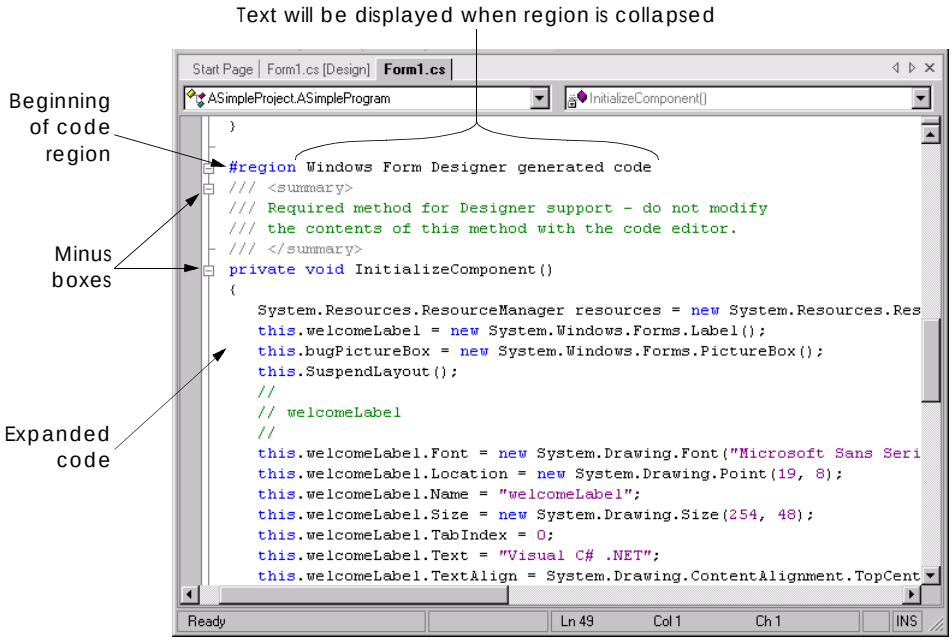

Fig. 4.17 Windows Form Designer generated code when expanded.

When we created this application, we used the **Properties** window to set properties for the form, label and picture box. Once a property was set, the form or control was updated immediately. Forms and controls contain a set of *default properties*, which are displayed initially in the **Properties** window when a form or control is selected. These default properties provide the initial characteristics of a form or control when it is created. When a control, such as a label, is placed on the form, the IDE adds code to the class (e.g., **`ASimpleProgram`**) that creates the control and sets some of the control's property values, such as the name of the control and its location on the form. Figure 4.18 shows a portion of the code generated by the IDE for setting the label's (i.e., **`welcomeLabel`**'s) properties. These include the label's **`Font`**, **`Location`**, **`Name`**, **`Text`** and **`TextAlign`** properties.

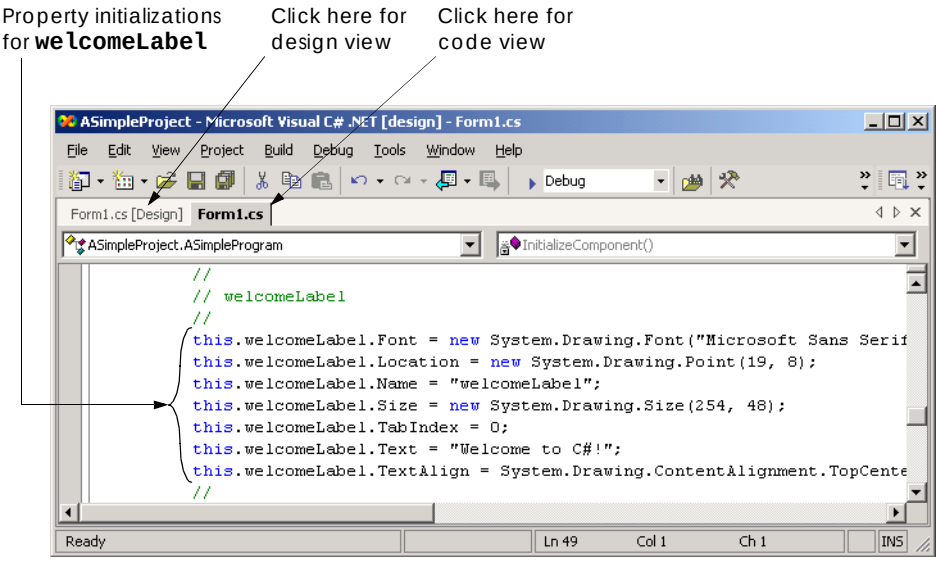

Fig. 4.18 Code generated by the IDE for `welcomeLabel`.

The values assigned to the properties are based on the values in the **Properties** window. We now demonstrate how the IDE updates the Windows Form Designer generated code it generates when a property value in the **Properties** window changes. During this process, we must switch between code view and design view. To switch views, select the corresponding tabs—**Form1.cs*** for code view and **Form1.cs* [Design]** for design view. Alternatively, the programmer can select **View > Code** or **View > Designer**. Perform the following steps:

1. *Modify the label control's* ***`Text`*** *property using the* ***Properties*** *window*. Recall that properties can be changed in design view by clicking a form or control to select it, then modifying the appropriate property in the **Properties** window. Change the **`Text`** property of the label to "**`Deitel`**" (Fig. 4.19).

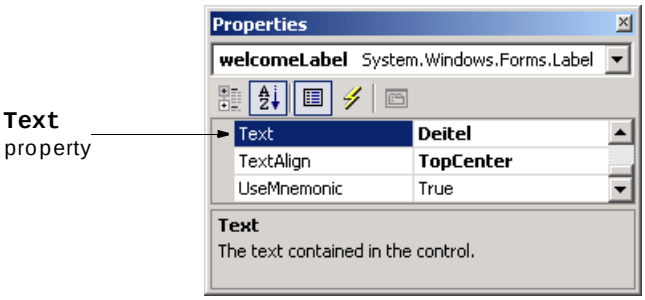

Fig. 4.19 Using the **Properties** window to set a property value.

2. *Examine the changes in the code view*. Switch to code view (**View > Code**) and examine the code. Notice that the label's **`Text`** property is now assigned the text that we entered in the **Properties** window (Fig. 4.20). When a property is changed in design mode, the Windows Form Designer updates the appropriate line of code in the class to reflect the new value.

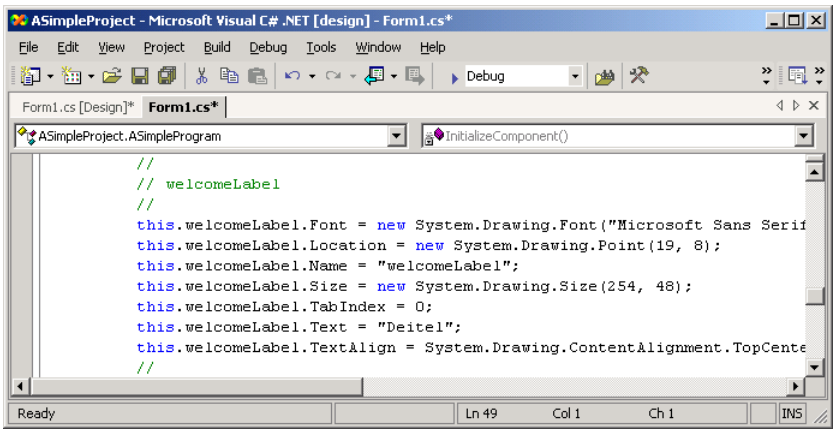

Fig. 4.20 Windows Form Designer generated code reflecting new property values.

3. *Modifying a property value in code view*. In the code view editor, locate the three lines of comments indicating the initialization for **`welcomeLabel`** and change the **`string`** assigned to **`this.welcomeLabel.Text`** from "**`Deitel`**" to "**`Visual C# .NET`**" (Fig. 4.21). Now, switch to design mode (**View > Designer**). The label now displays the updated text, and the **Properties** window for **`welcomeLabel`** displays the new **`Text`** value (Fig. 4.22). [*Note*: Property values should not be set using the techniques presented in this step. Here, we modify the property value in the IDE generated code only as a demonstration of the relationship between program code and the Windows Form Designer.]

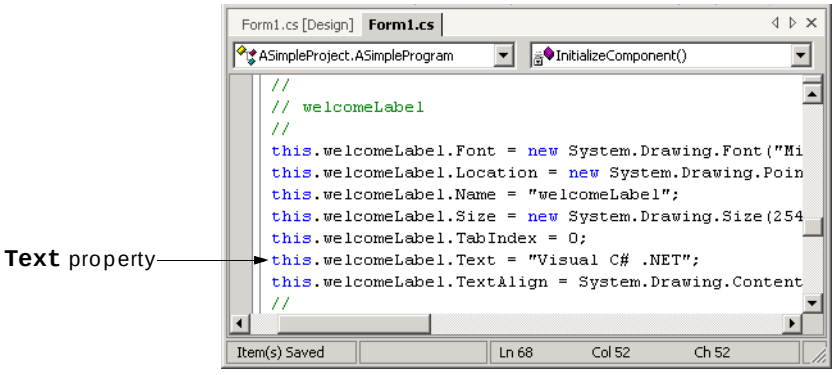

Fig. 4.21 Changing a property in the code view editor.

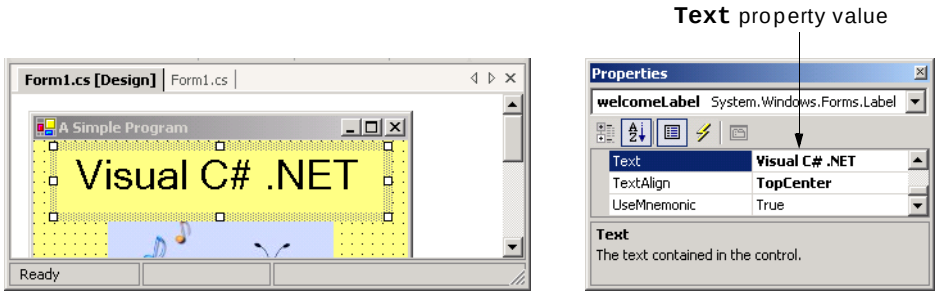

Fig. 4.22 New **`Text`** property value reflected in design mode.

4. *Change the label's* ***`Text`*** *property at runtime.* In the previous steps, we set properties at design time. Often, however, it is necessary to modify a property while a program is running. For example, to display the result of a calculation, a label's text can be assigned a **`string`** containing the result. In console applications, such code is located in **`Main`**. In Windows applications, we must create a method that executes when the form is loaded into memory during program execution. Like **`Main`**, this method is invoked when the program is run. Double-clicking the form in design view adds a method named ***`ASimpleProgram_Load`*** to the class (Fig. 4.23). The cursor is placed in the body of the **`ASimpleProgram_Load`** method definition. Notice that **`ASimpleProgram_Load`** is not part of the Windows Form Designer generated code. Add the statement **`welcomeLabel.Text = "C#";`** in the body of the method definition (Fig. 4.23). In C#, properties are accessed by placing the property name (i.e., **`Text`**) after the object name (i.e., **`welcomeLabel`**), separated by a dot operator. This syntax is similar to that used when accessing object methods. Notice that the *IntelliSense* feature displays the **`Text`** property in the member list after the class name and dot operator have been typed (Fig. 4.24).

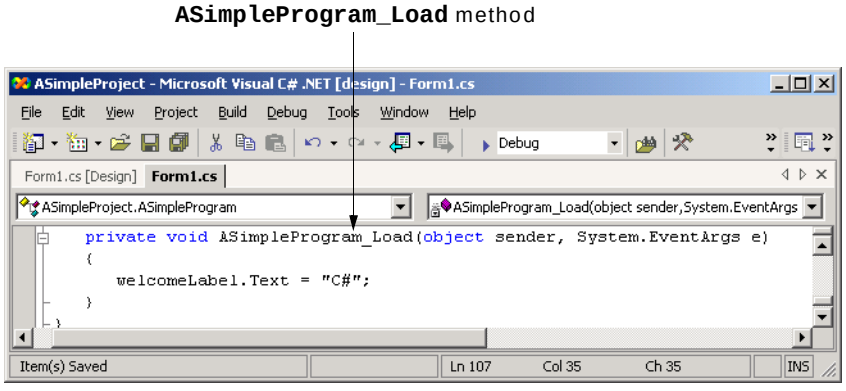

Fig. 4.23 Method **`ASimpleProgram_Load`**.

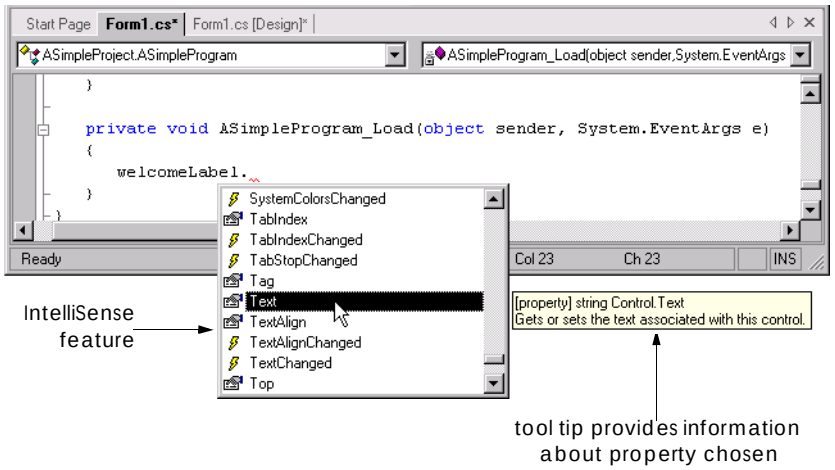

Fig. 4.24 IntelliSense feature of Visual Studio .NET.

5. *Examine the results of the* ***ASimpleProgram_Load*** *method.* Notice that the text in the label looks the same in **Design** mode as it did in Fig. 4.22. Note also that the property window still displays the value "**Visual C# .NET**" as the label's **Text** property and that the IDE generated code has not changed either. Select **Build > Build** then **Debug > Start** to run the program. Once the form is displayed, the text in the label reflects the property assignment in **ASimpleProgram_Load** (Fig. 4.25).

Fig. 4.25 Result of changing a property value at runtime.

6. *Terminate program execution.* Click the close button to terminate program execution. Once again, notice that both the label and the label's **Text** property contain the text **Visual C# .NET**. The IDE generated code also contains the text **Visual C# .NET**, which is assigned to the label's **Text** property.