Thanks for downloading this chapter from *Murach’s C++ 2008*. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its “how-to” headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our [website](#). From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on .NET development.

Thanks for your interest in our books!
Contents

Introduction

**Section 1** An introduction to Visual Studio

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>How to get started with Visual Studio</td>
<td>3</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>How to design a Windows Forms application</td>
<td>35</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>How to code and test a Windows Forms application</td>
<td>57</td>
</tr>
</tbody>
</table>

**Section 2** The language essentials

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 4</td>
<td>How to work with numeric and string data</td>
<td>89</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>How to code control structures</td>
<td>125</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>How to code methods and event handlers</td>
<td>151</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>How to handle exceptions and validate data</td>
<td>171</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>How to work with arrays and collections</td>
<td>199</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>How to work with dates and strings</td>
<td>239</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>More skills for working with Windows forms and controls</td>
<td>265</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>How to debug an application</td>
<td>307</td>
</tr>
</tbody>
</table>

**Section 3** Object-oriented programming with C++

<table>
<thead>
<tr>
<th>Chapter 12</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 13</td>
<td>How to create and use classes</td>
<td>333</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>How to work with indexers, delegates, events, and operators</td>
<td>367</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>How to work with inheritance</td>
<td>395</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>How to work with interfaces and generics</td>
<td>433</td>
</tr>
</tbody>
</table>

**Section 4** Other skills for C++ developers

<table>
<thead>
<tr>
<th>Chapter 17</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 18</td>
<td>How to work with files and data streams</td>
<td>477</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>How to work with XML files</td>
<td>499</td>
</tr>
</tbody>
</table>

**Reference aids**

<table>
<thead>
<tr>
<th>Appendix A</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>How to install and use the software and files for this book</td>
<td>551</td>
</tr>
</tbody>
</table>
Chapter 3
How to code and test a Windows Forms application

In the last chapter, you learned how to design a form for a Windows Forms application. In this chapter, you’ll learn how to code and test a Windows Forms application. When you’re done, you’ll be able to develop simple applications of your own.

An introduction to coding.................................................... 58
Introduction to object-oriented programming .................................................... 58
How to refer to properties, methods, and events .......................................... 60
How an application responds to events .................................................. 62
How to add code to a form .................................................................. 64
How to create an event handler for the default event of a form or control ...... 64
How to delete an event handler .............................................................. 64
The event handlers for the Invoice Total form .............................................. 66
More coding skills ................................................................ 68
How to code with a readable style .......................................................... 68
How to code comments .............................................................................. 70
How to work with the Text Editor toolbar .................................................. 72
How to collapse or expand blocks of code .................................................. 72
How to get help information .................................................................. 74
How to run, test, and debug a project ................................................. 76
How to run a project .............................................................................. 76
How to detect and correct syntax errors .................................................. 78
How to test a project .............................................................................. 80
How to debug runtime errors .................................................................. 82
Perspective............................................................................ 84
An introduction to coding

Before you learn the mechanics of adding code to a form, it’s important to understand some of the concepts behind object-oriented programming.

Introduction to object-oriented programming

Whether you know it or not, you are using object-oriented programming as you design a Windows form with Visual Studio’s Form Designer. That’s because each control on a form is an object, and the form itself is an object. These objects are derived from classes that are part of the .NET Class Library.

When you start a new project from the Windows Forms Application template, you are actually creating a new class that inherits the characteristics of the Form class that’s part of the .NET Class Library. Later, when you run the form, you are actually creating an instance of your form class, and this instance is known as an object.

Similarly, when you add a control to a form, you are actually adding a control object to the form. Each control is an instance of a specific class. For example, a text box control is an object that is an instance of the TextBox class. Similarly, a label control is an object that is an instance of the Label class. This process of creating an object from a class can be called instantiation.

As you progress through this book, you will learn much more about classes and objects because C++ is an object-oriented language. In chapter 12, for example, you’ll learn how to use the C++ language to create your own classes. At that point, you’ll start to understand what’s actually happening as you work with classes and objects. For now, though, you just need to get comfortable with the terms and accept the fact that a lot is going on behind the scenes as you design a form and its controls.

Figure 3-1 summarizes what I’ve just said about classes and objects. It also introduces you to the properties, methods, and events that are defined by classes and used by objects. As you’ve already seen, the properties of an object define the object’s characteristics and data. For instance, the Name property gives a name to a control, and the Text property determines the text that is displayed within the control. In contrast, the methods of an object determine the operations that can be performed by the object.

An object’s events are signals sent by the object to your application that something has happened that can be responded to. For example, a Button control object generates an event called Click if the user clicks the button. Then, your application can respond by running a C++ method to handle the Click event.

By the way, the properties, methods, and events of an object or class are called the members of the object or class. You’ll learn more about properties, methods, and events in the next three figures.
A form object and its ten control objects

![Invoice Total form](image)

Class and object concepts

- An object is a self-contained unit that combines code and data. Two examples of objects you have already worked with are forms and controls.
- A class is the code that defines the characteristics of an object. You can think of a class as a template for an object.
- An object is an instance of a class, and the process of creating an object from a class is called instantiation.
- More than one object instance can be created from a single class. For example, a form can have several button objects, all instantiated from the same Button class. Each is a separate object, but all share the characteristics of the Button class.

Property, method, and event concepts

- Properties define the characteristics of an object and the data associated with an object.
- Methods are the operations that an object can perform.
- Events are signals sent by an object to the application telling it that something has happened that can be responded to.
- Properties, methods, and events can be referred to as members of an object.
- If you instantiate two or more instances of the same class, all of the objects have the same properties, methods, and events. However, the values assigned to the properties can vary from one instance to another.

Objects and forms

- When you use the Form Designer, Visual Studio automatically generates C++ code that creates a new class based on the Form class. Then, when you run the project, a form object is instantiated from the new class.
- When you add a control to a form, Visual Studio automatically generates C++ code in the class for the form that instantiates a control object from the appropriate class and sets the control’s default properties. When you move and size a control, Visual Studio automatically sets the properties that specify the location and size of the control.
How to refer to properties, methods, and events

As you enter the code for a form in the Code Editor window, you often need to refer to the properties, methods, and events of the form’s objects. To do that, you type the name of the object, followed by \texttt{\textasciitilde} (also known as an \textit{arrow operator, or arrow}) and the name of the member. This is summarized in figure 3-2.

In addition to referring to the properties, methods, and events of an object, you can refer to some of the properties and methods of a class directly from that class. To do that, you type the name of the class followed by \texttt{::} (also known as the \textit{scope resolution operator}) and the name of the member. A property or method that you can refer to directly from a class like this is called a static member. You’ll learn more about static members in chapter 4.

To make it easier for you to refer to the members of an object or class, Visual Studio’s IntelliSense feature displays a list of the members that are available for that object or class after you type the object or class name followed by the \texttt{\textasciitilde} or \texttt{::} operator. Then, you can highlight the entry you want by clicking on it, typing one or more letters of its name, or using the arrow keys to scroll through the list. In most cases, you can then complete the entry by pressing the Tab or Enter key.

To give you an idea of how properties, methods, and events are used in code, this figure shows examples of each. In the first example for properties, code is used to set the value that’s displayed for a text box to 10. In the second example, code is used to set a text box’s ReadOnly property to true. Although you can also use the Properties window to set these values, that just sets the properties at the start of the application. By using code, you can change the properties as an application is running.

In the first example for methods, the Focus method of a text box is used to move the focus to that text box. In the second example, the Close method of a form is used to close the active form. In this example, the \texttt{this} keyword is used instead of the name of the form. Here, \texttt{this} refers to the current instance of the active form. In the third example, the ToDouble method of the Convert class is used to convert the string value in a text box to a double value. You’ll learn more about converting values like this in chapter 4.

As you progress through this book, you’ll learn how to use the methods for many types of objects, and you’ll learn how to supply arguments within the parentheses of a method. For now, though, just try to understand that you can call a method from a class or an object and that you must code a set of parentheses after the method name whether or not it requires arguments.

Although you’ll frequently refer to properties and methods as you code an application, you’ll rarely need to refer to an event. That’s because Visual Studio automatically generates the code for working with events, as you’ll see later in this chapter. To help you understand the code that Visual Studio generates, however, the last example in this figure shows how you refer to an event. In this case, the code refers to the Click event of a button named btnExit.
A member list that’s displayed in the Code Editor window

![Image of Code Editor window]

The syntax for referring to a member of an object or class

```plaintext
objectName->MemberName
ClassName::MemberName
```

Statements that refer to properties

```plaintext
txtTotal->Text = "10";
Assigns a string holding the number 10 to the Text property of the text box named txtTotal.

txtTotal->ReadOnly = true;
Assigns the true value to the ReadOnly property of the text box named txtTotal so the user can’t change its contents.
```

Statements that refer to methods

```plaintext
txtMonthlyInvestment->Focus();
Uses the Focus method to move the focus to the text box named txtMonthlyInvestment.

this->Close();
Uses the Close method to close the form that contains the statement. In this example, this is a keyword that is used to refer to the current instance of the class.

Convert::ToDouble(txtSubTotal->Text);
Uses the ToDouble method of the Convert class to convert a string value to a double value.
```

Code that refers to an event

```plaintext
btnExit->Click
Refers to the Click event of a button named btnExit.
```

How to enter member names when working in the Code Editor

- To display a list of the available members for a class, type the class name followed by a double colon (::). For an object, type the name followed by an arrow (->). Then, you can type one or more letters of the member name, and the Code Editor will select the first entry in the list that matches those letters. Or, you can scroll down the list to select the member you want. Once it’s selected, press the Tab or Enter key to insert the member into your code.

- If a member list isn’t displayed, select the Tools→Options command to display the Options dialog box. Then, expand the Text Editor group, select the C++ category, and check the Auto List Members and Parameters Information boxes.

Figure 3-2 How to refer to properties, methods, and events
How an application responds to events

Windows Forms applications are event-driven. That means they work by responding to the events that occur on objects. To respond to an event, you code a special type of method known as an event handler. When you do that, Visual Studio generates a statement that connects, or wires, the event handler to the event. This is called event wiring, and it’s illustrated in figure 3-3.

In this figure, the user clicks the Exit button on the Invoice Total form. Then, Visual Studio uses the statement it generated to wire the event to determine what event handler to execute in response to the event. In this case, the btnExit->Click event is wired to the method named btnExit_Click, so this method is executed. As you can see, this event handler contains a single statement that uses the Close method to close the form.

This figure also lists some common events for controls and forms. One control event you’ll respond to frequently is the Click event. This event occurs when the user clicks an object with the mouse. Similarly, the DoubleClick event occurs when the user double-clicks an object.

Although the Click and DoubleClick events are started by user actions, that’s not always the case. For instance, the Enter and Leave events typically occur when the user moves the focus to or from a control, but they can also occur when code moves the focus to or from a control. Similarly, the Load event of a form occurs when a form is loaded into memory. For the first form of an application, this typically happens when the user starts the application. And the Closed event occurs when a form is closed. For the Invoice Total form presented in this figure, this happens when the user selects the Exit button or the Close button in the upper right corner of the form.

In addition to the events shown here, most objects have many more events that the application can respond to. For example, events occur when the user positions the mouse over an object or when the user presses or releases a key. However, you don’t typically respond to those events.
Event: The user clicks the Exit button

Wiring: The application determines what method to execute

```csharp
this->btnExit->Click += gcnew System::EventHandler(this,
    &frmInvoiceTotal::btnExit_Click);
```

Response: The method for the Click event of the Exit button is executed

```csharp
private: System::Void btnExit_Click(System::Object^ sender,
    System::EventArgs^ e)
{
    this->Close();
}
```

Common control events

<table>
<thead>
<tr>
<th>Event</th>
<th>Occurs when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click</td>
<td>...the user clicks the control.</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>...the user double-clicks the control.</td>
</tr>
<tr>
<td>Enter</td>
<td>...the focus is moved to the control.</td>
</tr>
<tr>
<td>Leave</td>
<td>...the focus is moved from the control.</td>
</tr>
</tbody>
</table>

Common form events

<table>
<thead>
<tr>
<th>Event</th>
<th>Occurs when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>...the form is loaded into memory.</td>
</tr>
<tr>
<td>Closing</td>
<td>...the form is closing.</td>
</tr>
<tr>
<td>Closed</td>
<td>...the form is closed.</td>
</tr>
</tbody>
</table>

Concepts

- Windows Forms applications work by responding to events that occur on objects.
- To indicate how an application should respond to an event, you code an event handler, which is a special type of method that handles the event.
- To connect the event handler to the event, Visual Studio automatically generates a statement that wires the event to the event handler. This is known as event wiring.
- An event can be an action that’s initiated by the user like the Click event, or it can be an action initiated by program code like the Closed event.
How to add code to a form

Now that you understand some of the concepts behind object-oriented coding, you’re ready to learn how to add code to a form. Because you’ll learn the essentials of the C++ language in the chapters that follow, though, I won’t focus on the coding details right now. Instead, I’ll focus on the concepts and mechanics of adding the code to a form.

How to create an event handler for the default event of a form or control

Although you can create an event handler for any event of any object, you’re most likely to create event handlers for the default events of forms and controls. So that’s what you’ll learn to do in this chapter. Then, in chapter 6, you’ll learn how to create event handlers for other events.

To create an event handler for the default event of a form or control, you double-click the object in the Form Designer. Then, Visual Studio opens the Code Editor, generates a method declaration for the default event of the object, and places the insertion point on a blank line between the opening and closing braces of that declaration. As a result, you can immediately start typing the C++ statements that you want to include in the body of the method.

To illustrate, figure 3-4 shows the code that was generated when I double-clicked the Calculate button on the Invoice Total form. In this figure, the code for the form is stored in a file named frmInvoiceTotal.h. In addition, the name of the method is the name of the object (btnCalculate), an underscore, and the name of the event (Click). The statement that wires the Click event of this button to this event handler is stored within the Windows Form Designer code region of the same file.

Before you start an event handler for a control, you should set the Name property of the control as described in chapter 2. That way, this name will be reflected in the method name of the event handler as shown in this figure. If you change the control name after starting an event handler for it, Visual Studio will change the name of the object in the event wiring, but it won’t change the name of the object in the method name. And that can be confusing when you’re first learning C++.

You should also avoid modifying the method declaration that’s generated for you when you create an event handler. In chapter 6, you’ll learn how to modify the method declaration. But for now, you should leave the method declaration alone and focus on adding code within the body of the method.

How to delete an event handler

If you add an event handler by mistake, you can’t just delete it. If you do, you’ll get errors when you try to run the application. These errors will be
The method that handles the Click event of the Calculate button

How to handle the Click event of a button

1. In the Form Designer, double-click the control. This opens the Code Editor, generates the declaration for the method that handles the event, and places the cursor within this declaration.

2. Type the C++ code between the opening brace ( { ) and the closing brace ( } ) of the method declaration.

3. When you are finished writing code, you can return to the Form Designer by clicking the View Designer button in the Solution Explorer window.

How to handle the Load event for a form

- Follow the procedure shown above, but double-click the form itself.

Description

- The method declaration for the event handler that’s generated when you double-click on an object in the Form Designer includes a method name that consists of the object name, an underscore, and the event name.

- The event handler is inserted at the end the header file for the form.

- The code that’s generated when you design a form, including the statement that wires the event to the event handler, is inserted at the beginning of the header file for the form. If necessary, you can open this file to view or delete the event wiring.

- In chapter 6, you’ll learn how to handle events other than the default event.

Figure 3-4 How to create an event handler for the default event of a form or control
displayed in an Error List window as shown in figure 3-11, and they will indicate that the event handler is missing.

That’s because when you create an event handler, Visual Studio also generates a statement that wires the event to the event handler. As a result, if you delete an event handler, you must also delete the wiring for the event. The easiest way to do that is to double-click on one of the error messages in the Error List window. This will jump to the statement that contains the wiring for the missing event handler. Then, you can delete this statement.

### The event handlers for the Invoice Total form

Figure 3-5 presents the two event handlers for the Invoice Total form. The code that’s shaded in this example is the code that’s generated when you double-click the Calculate and Exit buttons in the Form Designer. You have to enter the rest of the code yourself.

I’ll describe this code briefly here so you have a general idea of how it works. If you’re new to programming, however, you may not understand the code completely until after you read the next two chapters.

The event handler for the Click event of the Calculate button calculates the discount percent, discount amount, and invoice total based on the subtotal entered by the user. Then, it displays those calculations in the appropriate text box controls. For example, if the user enters a subtotal of $1000, the discount percent will be 20%, the discount amount will be $200, and the invoice total will be $800.

In contrast, the event handler for the Click event of the Exit button contains just one statement that executes the Close method of the form. As a result, when the user clicks this button, the form is closed, and the application ends.

In addition to the code that’s generated when you double-click the Calculate and Exit buttons, Visual Studio generates other code that implements the form and controls that you designed in the Form Designer. You can see this code at the beginning of the header file for the form. Although you may want to look at this code to see how it works, you shouldn’t modify this code with the Code Editor as it may cause problems with the Form Designer. The one exception is deleting unnecessary event wiring statements.

When you enter C++ code, you must be aware of the coding rules summarized in this figure. In particular, note that each method contains a block of code that’s enclosed in braces. As you’ll see throughout this book, braces are used frequently in C++ to identify blocks of code. Also, note that each statement ends with a semicolon. This is true even if the statement spans several lines of code.

You should also realize that C++ is a case-sensitive language. As a result, you must use exact capitalization for all C++ keywords, class names, object names, variable names, and so on.
The event handlers for the Invoice Total form

```csharp
private: System::Void btnCalculate_Click(System::Object^ sender, 
                                     System::EventArgs^ e)
{
    double subtotal = Convert::ToDouble(txtSubtotal->Text);
    double discountPercent = 0;
    if (subtotal >= 500)
    {
        discountPercent = .2;
    }
    else if (subtotal >= 250 && subtotal < 500)
    {
        discountPercent = .15;
    }
    else if (subtotal >= 100 && subtotal < 250)
    {
        discountPercent = .1;
    }
    double discountAmount = subtotal * discountPercent;
    double invoiceTotal = subtotal - discountAmount;
    txtDiscountPercent->Text = discountPercent.ToString("p1");
    txtDiscountAmount->Text = discountAmount.ToString("c");
    txtTotal->Text = invoiceTotal.ToString("c");
    txtSubtotal->Focus();
}

private: System::Void btnExit_Click(System::Object^ sender, 
                                       System::EventArgs^ e)
{
    this->Close();
}
```

Coding rules

- Use spaces to separate the words in each statement.
- Use exact capitalization for all keywords, class names, object names, variable names, etc.
- End each statement with a semicolon.
- Each block of code must be enclosed in braces ({}). That includes the block of code that defines the body of a method.

Description

- When you double-click the Calculate and Exit buttons in the Form Designer, it generates the shaded code shown above. Then, you can enter the rest of the code within the event handlers.
- The first event handler for the Invoice Total form is executed when the user clicks the Calculate button. This method calculates and displays the discount percent, discount amount, and total based on the subtotal entered by the user.
- The second event handler for the Invoice Total form is executed when the user clicks the Exit button. This method closes the form, which ends the application.

Figure 3-5 The event handlers for the Invoice Total form
More coding skills

At this point, you should understand the mechanics of adding code to a form. To code effectively, however, you’ll need some additional skills. The topics that follow present some of the most useful coding skills.

How to code with a readable style

In figure 3-5, you learned some coding rules that you must follow when you enter the code for an application. If you don’t, Visual Studio reports syntax errors that you have to correct before you can continue. You’ll see how that works in figure 3-11.

Besides adhering to the coding rules, though, you should try to write your code so it’s easy to read, debug, and maintain. That’s important for you, but it’s even more important if someone else has to take over the maintenance of your code. You can create more readable code by following the three coding recommendations presented in figure 3-6.

To illustrate, this figure presents two versions of an event handler. Both versions accomplish the same task. As you can see, however, the first one is easier to read than the second one because it follows our coding recommendations.

The first coding recommendation is to use indentation and extra spaces to align related elements in your code. This is possible because you can use one or more spaces, tabs, or returns to separate the elements in a C++ statement. In this example, all of the statements within the event handler are indented. In addition, the if-else statements are indented and aligned so you can easily identify the parts of this statement.

The second recommendation is to separate the words, values, and operators in each statement with spaces. In the unreadable code in this figure, for example, you can see that each line of code except for the method declaration includes at least one operator. Because the operators aren’t separated from the word or value on each side of the operator, the code is difficult to read. In contrast, the readable code includes a space on both sides of each operator.

The third recommendation is to use blank lines before and after groups of related statements to set them off from the rest of the code. This too is illustrated by the first method in this figure. Here, the code is separated into five groups of statements. In a short method like this one, this isn’t too important, but it can make a long method much easier to follow.

Throughout this chapter and book, you’ll see code that illustrates the use of these recommendations. You will also receive other coding recommendations that will help you write code that is easy to read, debug, and maintain.

As you enter code, the Code Editor will automatically assist you in formatting your code. When you press the Enter key at the end of a statement, for example, the Editor will indent the next statement to the same level. Although you can change how this works using the Options dialog box, you probably won’t want to do that.
A method written in a readable style

```csharp
private: System::Void btnCalculate_Click(System::Object^ sender,
                                         System::EventArgs^ e)
{
    double subtotal = Convert::ToDouble(txtSubtotal->Text);
    double discountPercent = 0;

    if (subtotal >= 500)
    {
        discountPercent = .2;
    }
    else if (subtotal >= 250 && subtotal < 500)
    {
        discountPercent = .15;
    }
    else if (subtotal >= 100 && subtotal < 250)
    {
        discountPercent = .1;
    }

    double discountAmount = subtotal * discountPercent;
    double invoiceTotal = subtotal - discountAmount;

    txtDiscountPercent->Text = discountPercent.ToString("p1");
    txtDiscountAmount->Text = discountAmount.ToString("c");
    txtTotal->Text = invoiceTotal.ToString("c");

    txtSubtotal->Focus();
}
```

A method written in an unreadable style

```csharp
private: System::Void btnCalculate_Click(System::Object^ sender, System::EventArgs^ e){
    double subtotal = Convert::ToDouble(txtSubtotal->Text);
    double discountPercent=0;
    if (subtotal>=500) discountPercent=.2;
    else if (subtotal>=250&&subtotal<500) discountPercent=.15;
    else if (subtotal>=100&&subtotal<250) discountPercent=.1;
    double discountAmount=subtotal*discountPercent;
    double invoiceTotal=subtotal-discountAmount;

    txtDiscountPercent->Text=discountPercent.ToString("p1");
    txtDiscountAmount->Text=discountAmount.ToString("c");
    txtTotal->Text=invoiceTotal.ToString("c");

    txtSubtotal->Focus();
}
```

Coding recommendations

- Use indentation and extra spaces to align statements and blocks of code so they reflect the structure of the program.
- Use spaces to separate the words, operators, and values in each statement.
- Use blank lines before and after groups of related statements.

Note

- As you enter code in the Code Editor, Visual Studio automatically adjusts its formatting by default.

Figure 3-6 How to code with a readable style
How to code comments

Comments are used to document what the program does and what specific blocks and lines of code do. Since the C++ compiler ignores comments, you can include them anywhere in a program without affecting your code. Figure 3-7 shows you how to code two types of comments.

First, this figure shows a delimited comment at the start of a method. This type of comment is typically used to document information that applies to an entire method or to any other large block of code. You can include any useful or helpful information in a delimited comment such as a general description of the block, the author’s name, the completion date, the files used by the block, and so on.

To document the purpose of a single line of code or a block of code, you can use single-line comments. Once the compiler reads the slashes (//) that start this type of comment, it ignores all characters until the end of the current line. In this figure, single-line comments have been used to describe each group of statements. In addition, single-line comments have been used at the end of some lines of code to clarify the code.

Although many programmers sprinkle their code with comments, that shouldn’t be necessary if you write your code so it’s easy to read and understand. Instead, you should use comments only to clarify code that’s difficult to understand. The trick, of course, is to provide comments for the code that needs explanation without cluttering the code with unnecessary comments. For example, an experienced C++ programmer wouldn’t need any of the comments shown in this figure.

One problem with comments is that they may not accurately represent what the code does. This often happens when a programmer changes the code, but doesn’t change the comments that go along with it. Then, it’s even harder to understand the code, because the comments are misleading. So if you change the code that has comments, be sure to change the comments too.

Incidentally, all comments are displayed in the Code Editor in green by default, which is different from the color of the words in the C++ statements. That makes it easy to identify the comments.
A method with comments

private: System::Void btnCalculate_Click(System::Object^ sender,
    System::EventArgs^ e)
{
    /*******************************
    * this method calculates the total
    * for an invoice depending on a
    * discount that's based on the subtotal
    ****************************************/

    // get the subtotal amount from the Subtotal text box
double subtotal = Convert::ToDouble(txtSubtotal->Text);

    // set the discountPercent variable based
    // on the value of the subtotal variable
    double discountPercent = 0;
    if (subtotal >= 500)
    {
        discountPercent = .2;
    }
    else if (subtotal >= 250 && subtotal < 500)
    {
        discountPercent = .15;
    }
    else if (subtotal >= 100 && subtotal < 250)
    {
        discountPercent = .1;
    }

    // calculate and assign the values for the
    // discountAmount and invoiceTotal variables
decimal discountAmount = subtotal * discountPercent;
decimal invoiceTotal   = subtotal - discountAmount;

    // format the values and display them in their text boxes
    txtDiscountPercent->Text =          // percent format
        discountPercent.ToString("p1");   // with 1 decimal place
    txtDiscountAmount->Text =
        discountAmount.ToString("c");    // currency format
    txtTotal->Text =
        invoiceTotal.ToString("c");

    // move the focus to the Subtotal text box
    txtSubtotal->Focus();
}

Description

- **Comments** are used to help document what a program does and what the code within it does.

- To code a **single-line comment**, type // before the comment. You can use this technique to add a comment on its own line or to add a comment at the end of a line.

- To code a **delimited comment**, type /* at the start of the comment and */ at the end. You can also code asterisks to identify the lines in the comment, but that isn’t necessary.
How to work with the Text Editor toolbar

Figure 3-8 shows how you can use the Text Editor toolbar to work with code. If you experiment with this toolbar, you’ll find that its buttons provide some useful functions for working with comments and indentation and for moving from one place to another.

In particular, you can use the Text Editor toolbar to modify several lines of code at once. For example, during testing, you can use this toolbar to comment out several lines of code by selecting the lines of code and then clicking on the Comment Out button. Then, you can test the program without those lines of code. If necessary, you can use the Uncomment button to restore those lines of code. Similarly, you can use the Increase Indent and Decrease Indent buttons to adjust the indentation for selected lines of code.

You can also use the Text Editor toolbar to work with bookmarks. After you use the Toggle Bookmark button to mark lines of code, you can easily move between the marked lines of code by using the Next and Previous buttons. Although you usually don’t need bookmarks when you’re working with simple applications like the one shown here, bookmarks can be helpful when you’re working with applications that contain more than a few pages of code.

If you experiment with the other buttons on the Text Editor toolbar, you’ll find that they provide IntelliSense features like the ones you learned about earlier in this chapter for referring to properties, methods, and events. You can use these buttons to force Visual Studio to display a member list or information about a member that’s displayed in the Code Editor.

How to collapse or expand blocks of code

As you write the code for an application, you may want to collapse or expand some of the regions, comments, and methods to make it easier to scroll through the code and locate specific sections of code. To do that, you can use the techniques described in figure 3-8. In this figure, for example, the Windows Form Designer generated code region has been collapsed so all you can see is a summary line.

You may also want to collapse or expand code before you print it. Then, in the dialog box for the File→Print command, you can check or uncheck the Hide Collapsed Regions box. If this box is checked, Visual Studio will only print the code that’s displayed in the Code Editor.
The Code Editor and the Text Editor toolbar

How to use the buttons of the Text Editor toolbar

- To display or hide the Text Editor toolbar, right-click in the toolbar area and choose Text Editor from the shortcut menu.
- To comment or uncomment several lines of code, select the lines and click the Comment Out or Uncomment button. During testing, you can comment out coding lines so they won’t be executed. That way, you can test new statements without deleting the old statements.
- To increase or decrease the indentation of several lines of code, select the lines and click the Increase Indent or Decrease Indent button. Or, press the Tab and Shift+Tab keys.
- To move quickly between lines of code, you can use the last eight buttons on the Text Editor toolbar to set and move between bookmarks.

How to collapse or expand regions of code

- If a region of code appears in the Code Editor with a minus sign (-) next to it, you can click the minus sign to collapse the region so just the first line is displayed.
- If a region of code appears in the Code Editor with a plus sign (+) next to it, you can click the plus sign to expand the region so all of it is displayed.
How to get help information

As you develop applications in C++, it’s likely that you’ll need some additional information about the IDE, the C++ language, an object, property, method, or event, or some other aspect of C++ programming. Figure 3-9 shows several ways you can get that information.

When you’re working in the Code Editor or the Form Designer, the quickest way to get help information is to press F1 while the insertion point is in a keyword or an object is selected. Then, Visual Studio opens a separate Help window like the one shown in this figure and displays the available information about the selected keyword or object. Another way to launch a Help window is to select a command from Visual Studio’s Help menu such as the Search, Contents, or Index command.

The Help window is split into two panes. The right pane shows the last help topic that you accessed. In this figure, for example, the right pane displays a help topic that provides information about working with the Code Editor.

The left pane, on the other hand, displays the Contents, Index, and Help Favorites tabs that help you locate help topics. In this figure, for example, the left pane displays the Index tab. At the top of this tab, the drop-down list has been used to filter help topics so they’re appropriate for C++ programmers. In addition, “code e” has been entered to navigate to the index entries that begin with those letters, and the Code Editor entry has been selected.

In addition to the topic that’s displayed in the right pane, all the topics that are available for a selected entry are listed in the Index Results window that’s displayed across the bottom of the screen. When the Code Editor entry was selected in this figure, for example, seven topics were listed in the Index Results window and the first topic was displayed by default. To display another topic, you simply click on it.

In the left pane, you can click on the Contents tab to display a list of help topics that are grouped by category. Or, you can click on the Help Favorites tab to view a list of your favorite help topics. At first, the Help Favorites tab won’t contain any help topics. However, you can add topics to this tab by displaying a topic and clicking on the Add To Help Favorites button that’s available from the toolbar.

You can display a Search tab in the right pane by clicking on the Search button in the toolbar. From this tab, you can enter a word or phrase to search for, along with the languages, technologies, and content you want to search. Then, when you click the Search button, the results are displayed in the tab and you can click a topic to display it.

When you display information in the Help window, you should realize that the Help window uses a built-in web browser to display help topics that are available from your computer and from the Internet. In addition, the Help window works much like a web browser. To jump to a related topic, you can click on a hyperlink. To move forward and backward through previously displayed topics, you can use the Forward and Back buttons. As a result, with a little practice, you shouldn’t have much trouble using this window.
The Help window

You can display a Help window by selecting an object in the Form Designer or positioning the insertion point in a keyword in the Code Editor and pressing F1.

You can also display a Help window by selecting a command (such as Index, Contents, or Search) from Visual Studio’s Help menu.

The Help window works like a web browser and can display help topics that are available from your computer or from the Internet. You can use the buttons in its toolbar to navigate between help topics or to add topics to your list of favorite topics.

The Help window is divided into two panes. The left pane displays the Contents, Index, and Help Favorites tabs that let you locate the help topics you want to display. The right pane displays each help topic in a separate window.

If you click on the Search button, the right pane will display a Search tab that lets you search for help topics by entering a word or phrase.

If you click on the How Do I button, the right pane will display a How Do I tab that lets you go to a topic by clicking on a link.

To close a tab, click on the Close button when the tab is active. To display a tab, click the tab or select it from the Active Files drop-down list that’s next to the Close button.
How to run, test, and debug a project

After you enter the code for a project, you are ready to run the project. If the project has changed since the last time you ran it, it’s first built so that any syntax errors are detected. Then, you can correct those errors and run the project again. When the project runs, you can test it to make sure it works the way you want it to, and you can debug it to remove any programming errors you find.

How to run a project

As you learned in chapter 1, you can run a project by clicking the Start Debugging button in the Standard toolbar, selecting the Start Debugging command from the Debug menu, or pressing the F5 key. This builds the project if it hasn’t been built already and causes the project’s main form to be displayed, as shown in figure 3-10. When you close this form, the application ends. Then, you’re returned to Visual Studio where you can continue working on your program.

You can also build a project without running it as described in this figure. In most cases, though, you’ll run the project so you can test and debug it.

If syntax errors are detected when you run a project, the errors are displayed in the Error List window. You can use this window to identify and correct the errors, as you’ll see in the next figure.
The form that's displayed when you run the Invoice Total project

![Invoice Total form]

**Description**

- To *run* a project, click the Start Debugging button in the Standard toolbar, select the Debug ➔ Start Debugging menu command, or press the F5 key. This causes Visual Studio to *build* the project and create an assembly. Then, assuming that there are no build errors, the assembly is run so the project’s form is displayed as shown above.

- By default, if you run a project and you’ve made changes to it since it was last built, a dialog box is displayed that asks if you want to build the project. If you select the Do Not Show This Dialog Again option, projects will always be built if they’ve been changed.

- If syntax errors (or build errors) are detected when a project is built, a dialog box that asks if you want to run the last successful build is displayed. In most cases, you won’t want to do that.

- To correct build errors, use the Error List window as shown in the next figure.

- You can build a project without running it by using the Build ➔ Build Solution command.

- When you build a project for the first time, all of the components of the project are built. After that, only the components that have changed are rebuilt. To rebuild all components whether or not they’ve changed, use the Build ➔ Rebuild Solution command.

---

Figure 3-10    How to run a project
How to detect and correct syntax errors

When Visual Studio builds a project, it starts by checking the syntax of each statement in your code. If any syntax errors, or build errors, are detected, they’re listed in the Error List window as shown in figure 3-11. Then, you can jump to an error in the Code Editor by double-clicking on it in the Error List window, and you can correct the error. When you do that, you should realize that the error will still be listed in the Error List window even after you’ve corrected it. The error isn’t cleared until you build the project again.

When you’re first getting started with C++, you will inevitably encounter a lot of errors. As a result, you may want to keep the Error List window open all the time. This makes it easy to see errors as soon as they occur. Then, once you get the hang of working with C++, you can conserve screen space by using the Auto Hide button so this window is only displayed when you point to the Error List tab at the lower edge of the screen.
The Code Editor and Error List windows with syntax errors displayed

Description

- Visual Studio checks the syntax of your C++ code when you build the project. If syntax errors are detected, they’re listed in the Error List window. Then, you can double-click on any error in the list to take you to its location in the Code Editor.

- If the Error List window isn’t open, you can display it by selecting the Other Windows/Error List command from the View menu. Then, if you want to hide this window, you can click its Auto Hide button.

- If the syntax errors aren’t displayed in the Error List window, you can click the button at the top of this window that indicates the number of errors to display the errors.

- After you’ve corrected all the errors, you can run the project again to rebuild it and clear the errors from the Error List window.

Figure 3-11  How to detect and correct syntax errors
How to test a project

When you test a project, you run it and make sure the application works correctly. As you test your project, you should try every possible combination of input data and user actions to be certain that the project works correctly in every case. Figure 3-12 provides an overview of the testing process for C++ applications.

To start, you should test the user interface. Make sure that each control is sized and positioned properly, that there are no spelling errors in any of the controls or in the form’s title bar, and that the navigation features such as the tab order and access keys work properly.

Next, subject your application to a carefully thought-out sequence of valid test data. Make sure you test every combination of data that the project will handle. If, for example, the project calculates the discount at different values based on the value of the subtotal, use subtotals that fall within each range.

Finally, test the program to make sure that it properly handles invalid data entered by users. For example, type text information into text boxes that expect numeric data. Leave fields blank. Use negative numbers where they shouldn’t be allowed. Remember that the goal of testing is to find all of the problems.

As you test your projects, you’ll eventually encounter runtime errors. These errors, also known as exceptions, occur when C++ encounters a problem that prevents a statement from being executed. If, for example, a user enters “ABC” into the Subtotal text box on the Invoice Total form, a runtime error will occur when the program tries to assign that value to a decimal variable.

When a runtime error occurs, Visual Studio displays an Unhandled Exception window like the one in this figure. Then, you can break into the debugger and use the debugging tools that you’ll be introduced to in the next figure to debug the error.

Runtime errors, though, should only occur when you’re testing a program. Before an application is put into production, it should be coded and tested so all runtime errors are caught by the application and appropriate messages are displayed to the user. You’ll learn how to do that in chapter 7 of this book.
Chapter 3  How to code and test a Windows Forms application  81

The message that’s displayed when a runtime error occurs

![Unhandled Exception window]

How to test a project

1. Test the user interface. Visually check all the controls to make sure they are displayed properly with the correct text. Use the Tab key to make sure the tab order is set correctly, verify that the access keys work right, and make sure that the Enter and Esc keys work properly.

2. Test valid input data. For example, enter data that you would expect a user to enter.

3. Test invalid data or unexpected user actions. For example, leave required fields blank, enter text data into numeric input fields, and use negative numbers where they are not appropriate. Try everything you can think of to make the program fail.

Description

- To test a project, you run the project to make sure it works properly no matter what combinations of valid or invalid data you enter or what sequence of controls you use.

- If a statement in your application can’t be executed, a runtime error, or exception, occurs. Then, if the exception isn’t handled by your application, the statement that caused the exception is highlighted and an Unhandled Exception window like the one above is displayed. At that point, you need to debug the application as explained in the next figure.

Figure 3-12  How to test a project
How to debug runtime errors

When a runtime error occurs, Visual Studio displays an Unhandled Exception window like the one you saw in figure 3-12. This window indicates the type of exception that occurred. If you click the Break button in this window, Visual Studio enters break mode. In this mode, Visual Studio displays the Code Editor, highlights the statement that couldn’t be executed, and displays the Debug toolbar.

Often, you can determine the cause of an exception (the bug) just by knowing the type of exception that occurred and what statement couldn’t be executed. But sometimes, it helps to find out what the current values in some of the variables or properties in the program are.

To do that, you can place the mouse pointer over a variable or the object a property is associated with in the code so a data tip is displayed for the variable or object as shown in figure 3-13. The data tip for a variable displays the current value of the variable. The data tip for an object displays the current values for all the members of the object. Then, you can locate the member to display its value. In this figure, the data tip for the Text property of the txtSubtotal control is “$100”, which shows that the user didn’t enter valid numeric data.

Once you find the cause of a bug, you can debug the application by adding code that prevents the exception from occurring again or handles the exception. Sometimes, you can do that in break mode and continue running the application. Often, though, you’ll exit from break mode before fixing the code. To exit, you can click the Stop Debugging button in the Debug toolbar. Then, you can correct the code and test the application again.

For now, don’t worry if you don’t know how to correct the problem in this example. Instead, you can assume that the user will enter valid data. In chapter 7, though, you’ll learn how to catch exceptions and validate all user entries for an application because that’s what a professional application has to do. And in chapter 11, you’ll learn a lot more about debugging.
How a project looks in break mode

**Description**

- When an application encounters a runtime error, you need to fix the error. This is commonly referred to as debugging, and the error is commonly referred to as a bug.

- When an application encounters a runtime error, it displays an Unhandled Exception window. The information in this window should give you an idea of what the error might be.

- You can click the Break button from the Unhandled Exception window to enter break mode. In break mode, the Debug toolbar is displayed along with other windows that provide debugging features.

- To display a data tip for a variable, move the mouse pointer over that variable in the Code Editor window. To display a data tip for a property, move the mouse pointer over its associated object and then find the property in the alphabetized member list.

- To exit break mode and end the application, click the Stop Debugging button in the Debug toolbar or press Shift+F5. Then, you can attempt to fix the error and run the application again.

- You’ll learn more about debugging in chapter 11.
Perspective

If you can code and test the Invoice Total project that’s presented in this chapter, you’ve already learned a lot about C++ programming. You know how to enter the code for the event handlers that make the user interface work the way you want it to. You know how to build and test a project. And you know some simple debugging techniques.

On the other hand, you’ve still got a lot to learn. For starters, you need to learn the C++ language. So in the next six chapters, you’ll learn the essentials of the C++ language. Then, in chapter 11, you’ll learn some debugging techniques that can be used with more advanced code.

Terms

- object-oriented programming
- object-oriented language
- object
- class
- instance
- instantiation
- property
- method
- event
- member
- arrow operator
- scope resolution operator
- static member
- event-driven application
- event handler
- event wiring
- method declaration
- method name
- statement
- block of code
- comment
- single-line comment
- delimited comment
- comment out a line
- bookmark
- collapse
- expand
- build a project
- run a project
- syntax error
- build error
- test a project
- runtime error
- exception
- bug
- debug
- break mode
- data tip

Exercise 3-1  Code and test the Invoice Total form

In this exercise, you’ll add code to the Invoice Total form that you designed in exercise 2-1. Then, you’ll build and test the project to be sure it works correctly. You’ll also experiment with debugging and review some help information.

Copy and open the Invoice Total application

1. Use the Windows Explorer to copy the Invoice Total project that you created for chapter 2 from the C:\C++ 2008\Chapter 02 directory to the C:\C++ 2008\Chapter 03 directory.
2. Open the Invoice Total solution (InvoiceTotal.sln) that’s now in the C:\C++\2008\Chapter 03\InvoiceTotal directory.

Add code to the form
3. Display the Invoice Total form in the Form Designer, and double-click on the Calculate button to open the Code Editor and generate the method declaration for the Click event of this object. Then, enter the code for this method as shown in figure 3-5.
4. Return to the Form Designer, and double-click the Exit button to generate the method declaration for the Click event of this object. Enter the statement shown in figure 3-5 for this event handler.

Test the application
5. Press F5 to build and run the project. If a dialog box is displayed that asks if you want to build the project, select the Do Not Show This Dialog Again option and then click the Yes button. The build should succeed and the Invoice Total form should appear. If not, you’ll need to use the Error List window as shown in figure 3-11 to correct the errors. Then, you can press F5 again to run the project.
6. Enter a valid numeric value in the first text box and click the Calculate button or press the Enter key to activate this button. Assuming that the calculation works, click the Exit button or press the Esc key to end the application. If either of these methods doesn’t work right, of course, you need to debug the problems and test the application again.

Enter invalid data and display data tips in break mode
7. Start the application again. This time, enter “xx” for the subtotal. Then, click the Calculate button. This will cause Visual Studio to display the Unhandled Exception window as shown in figure 3-12.
8. Note the statement pointed to by an arrow on the left and read the message that’s displayed in the Unhandled Exception window. Then, click the Break button in the Unhandled Exception window to close the window and enter break mode.
9. Move the mouse pointer over the variable in the statement that can’t be executed to display its value. Then, move the mouse pointer over the object associated with the property in this statement and locate the property to see its value. This shows that the code for this application needs to be enhanced so it checks for invalid data. You’ll learn how to do that in chapter 7. For now, though, click the Stop Debugging button in the Debug toolbar to end the application.

Create a syntax error and see how it affects the IDE
10. Next, change the name of the Subtotal text box from txtSubtotal to txtSubTotal. This will create an error since the capitalization doesn’t match the capitalization used by the Name property of the text box.
11. Try to run the application, and click No when Visual Studio tells you the build had errors and asks whether you want to continue with the last successful build.
12. Open the Error List window as described in figure 3-11. If necessary, click the button at the top of this window that indicates the number of errors to display the error. Then, double-click on the error, correct the statement that caused the error, and run the application again to make sure the problem is fixed.

**Generate and delete an event handler**

13. Display the Form Designer for the Invoice Total form and double-click a blank area on the form. This should generate an event handler for the Load event of the form.

14. Delete the event handler for the Load event of the form. Then, run the application. When you do, you’ll get a build error that indicates that the form does not contain a definition for this event handler.

15. Double-click on the error. This opens the header file for the form and jumps to the statement that wires the event handler. Delete this statement to correct the error.

16. If you’re curious, review the generated code that’s stored in the header file for this simple form. Then, click the minus sign to the left of the region named “Windows Form Designer generated code” to collapse this region.

17. Run the form to make sure it’s working correctly. When you return to the Code Editor, close the header file for the form.

**Experiment with the Help feature**

18. To see how context-sensitive help works, place the insertion point in the Focus method in the last statement of the first event handler and press F1. This should open a Help window that tells you more about this method.

19. In the left pane, select the Index tab to display the Index window. Type “format specifiers” into the Look For box in this window to see the entries that are listed under this topic. Next, if Visual C++ (or Visual C++ Express Edition) isn’t selected in the Filter By drop-down list, select it to show just the topics for C++. Then, click on one or more topics such as “numeric format strings” and “standard numeric format strings” to display them.

20. Continue experimenting with the Index, Contents, Help Favorites, and Search features to see how they work, and try using some of the buttons in the Web toolbar to see how they work. Then, close the Help window.

**Exit from Visual Studio**

21. Click the Close button for the Visual Studio window to exit from this application. If you did everything and got your application to work right, you’ve come a long way!
How to build your C++ programming skills

The easiest way is to let Murach’s C++ 2008 be your guide! So if you’ve enjoyed this chapter, I hope you’ll get your own copy of the book today. You can use it to:

- Teach yourself how to code C++ by using all the time- and work-saving features of Visual Studio 2008
- Learn how to compile, run, and enhance legacy C and native C++ code on the .NET platform
- Pick up a new skill whenever you want or need to by focusing on material that’s new to you
- Look up coding details or refresh your memory on forgotten details when you’re in the middle of developing a C++ application
- Loan to your colleagues who will be asking you more and more questions about C++ programming

To get your copy, you can order online at www.murach.com or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

You must be satisfied. Each book you buy directly from us must outperform any competing book or course you’ve ever tried, or send it back within 90 days for a full refund…no questions asked.

Thanks for your interest in Murach books!