

# HOUR 3



## Creating Our First ASP.NET Web Page

In the last two hours, we've spent quite a bit of time talking in very high-level terms about ASP.NET Web pages and the ASP.NET programming model. We've looked at how to configure our computer to serve ASP.NET Web pages, and we've looked at the role of the Web server. We've examined the HTML and source code portions of an ASP.NET Web page and looked at HTML controls and Web controls. We've created some very simple ASP.NET Web pages and have seen how to use the Web Matrix Project to create these pages.

In this hour we turn from these high-level discussions to actually building a useful ASP.NET Web page that illustrates the concepts discussed in the last two hours. Specifically, we'll be creating an ASP.NET Web page that serves as a financial calculator. This hour will focus on building the ASP.NET Web page, with only a light discussion of the source code and Web controls used. In the next hour, however, we will look at the ASP.NET Web page created in this hour in much more depth.

Fire up the Web Matrix Project and get ready to start creating your first practical ASP.NET Web page!

In this hour we will cover

- Creating the design requirements for the financial calculator
- Creating the user interface
- Adding the needed Web controls to the ASP.NET Web page
- Writing the code for the ASP.NET Web page's source code portion
- Testing the ASP.NET Web page

## Specifying the Design Requirements

Throughout this book we will be creating a number of ASP.NET Web pages, which involves creating both the ASP.NET Web page's HTML and its source code. When writing any piece of software, whether a Windows desktop application or a dynamic Web page, there are a number of development stages. First and foremost we need to decide what the purpose of the software is, along with what features and functionality the software should provide. After this we must sit down and actually write the software. Finally, we need to test the software and fix any bugs or errors that arise.

These three steps—design, development, and testing—should always be performed when creating an ASP.NET Web page, but too frequently, developers jump straight to the coding task without spending enough time in the planning stage. This initial planning stage, sometimes called the *design requirements* stage, is vital for the following reasons:

- It lays down a road map for the software project. Having a road map allows us to determine how much progress we've made at a given point, as well as how much we have left to accomplish.
- The design requirements spell out precisely what the software will provide.

To get into the habit, we will spend a bit of time discussing what features will be present and what user interface will be employed in the ASP.NET Web page we will be creating in this hour.



Without spending adequate time in the design requirements stage, you would be unable to accurately answer your boss when he asks, “How much longer will this take,” or “How much progress have you made?” Additionally, agreeing on a list of feature requirements—a task typically performed during the design requirements stage—avoids any confusion at the conclusion of the project; otherwise, your boss and client might wonder why a feature they thought was going to be present was not.

## Formulating the Features for Our Financial Calculator

An important step in the design requirements process is to list the features you plan on providing in your application. So far I have just mentioned that we will be creating a financial calculator, but let’s take the time to specifically define the features we want to provide.

For our financial calculator let’s build a loan calculator designed to determine the monthly payments for a fixed home *mortgage*. To determine the monthly payments required for a fixed mortgage, three inputs are needed:

1. The amount of money being borrowed (the principal)
2. The loan’s annual interest rate
3. The duration of the loan—typically 15 or 30 years (the loan’s term)

The output of our financial calculator, along with these three inputs, gives us the features of our financial calculator. In a sentence: Our financial calculator will compute the monthly payment of a fixed mortgage when provided the amount, duration, and interest rate of the mortgage.

## Deciding on the User Interface

After describing the features that the application will have, the next stage in the design requirements phase is to create a user interface. The user interface, or UI for short, is the means by which the user interacts with the application. How will the user enter these inputs? How will the results be displayed?

With large applications the user interface portion of the design requirements phase can take quite a while and be very involved. For our financial calculator, however, the user interface is fairly straightforward and will exist on a single Web page.

Essentially, our users need to be able to do two things: enter the three inputs discussed earlier and see the result of the calculation. These inputs can be entered via TextBox Web controls. The output of the financial calculator should show the mortgage's monthly cost.

Figure 3.1 contains a screenshot of the ASP.NET Web page financial calculator when first visited by the user. Note the three textboxes for the three inputs. Additionally, there is a button labeled Compute Monthly Cost that the user is instructed to click once having entered the required inputs.

**FIGURE 3.1**

*The user is asked to enter the three inputs.*

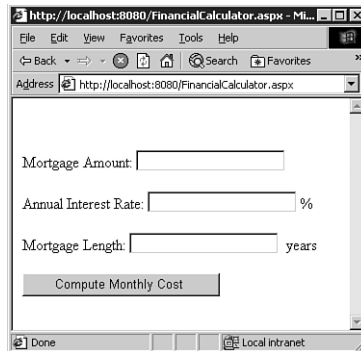
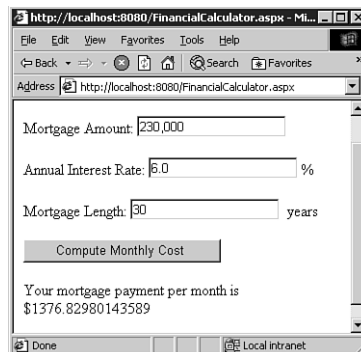


Figure 3.2 contains a screenshot of the financial calculator after the user has entered the requested inputs and has clicked the Compute Monthly Cost button. Note that the output shows how much money the mortgage will cost per month.

**FIGURE 3.2**

*The monthly cost of the mortgage is shown.*



In order to display the output of our calculation, we need to add a Label Web control to our ASP.NET page. This Label Web control will display the result of the calculation. Therefore, we should place this Label Web control in the ASP.NET Web page precisely where we want the final output to appear. As you can see from Figure 3.2, I have created the financial calculator so that the output appears below the input TextBoxes.

## Creating the User Interface

Now that we've completed the design requirements phase and have decided what features our financial calculator will provide, as well as how the interface will appear to the user, it's time to actually start creating our ASP.NET Web page.

The first task is to create the user interface (or UI), which is considered the HTML portion of our ASP.NET Web page. To construct this UI, we'll add a TextBox Web control for each of the three inputs, as well as a Button Web control that, when clicked, will perform the necessary computations.



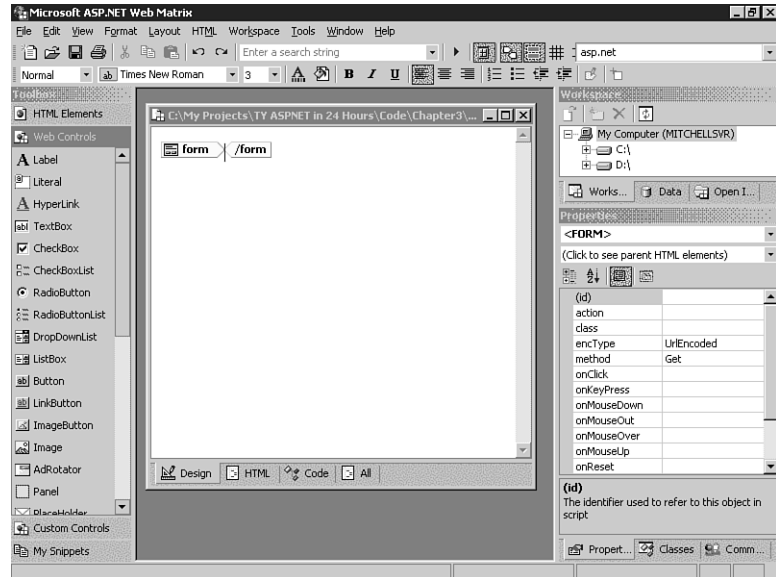
After creating the user interface, we will turn our attention to writing the needed source code to perform the financial computations.

To start creating our user interface, launch the Web Matrix Project and create a new ASP.NET Web page named `FinancialCalculator.aspx`, making certain to use the Visual Basic .NET Language option (the default). Before we add any content to the HTML portion of our ASP.NET Web page, first take a moment to turn on *glyphs*, which are markers in the designer that indicate the location of HTML controls and Web controls. To turn on glyphs, go to the View menu and choose the Glyphs option.

With glyphs activated, you should see, in your Design tab, two yellow tags that mark the beginning and end of your form HTML control. Figure 3.3 contains a screenshot of the Web Matrix Project with glyphs enabled.

**FIGURE 3.3**

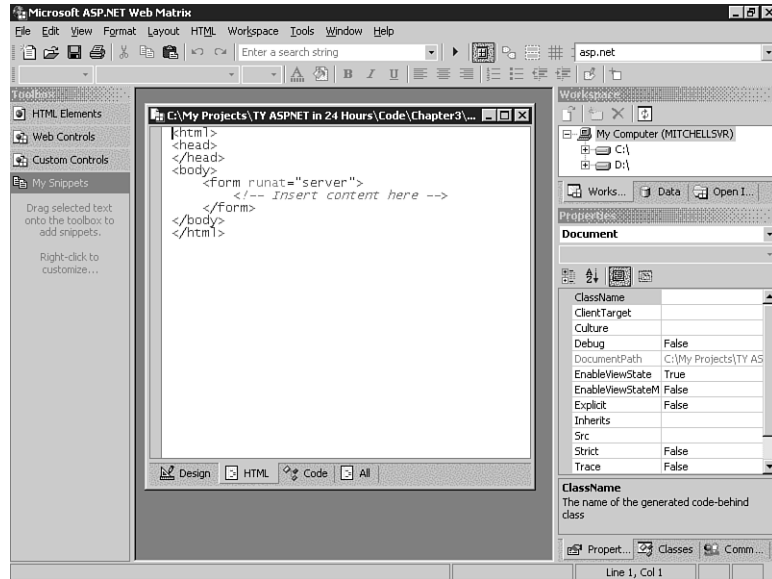
*Turning on glyphs uses markers to display invisible HTML controls and Web controls.*



If you do not see a Design tab for your ASP.NET Web page and instead see only Source and Preview tabs, you have the Web Matrix Project in Preview Mode and need to change to Design Mode. Refer back to the “Editing the HTML Content by Using the HTML Tab” section from Hour 2, “Understanding the ASP.NET Programming Model,” for information on switching back to Design Mode.

Recall from our discussion in the previous hour that the Web Matrix Project automatically inserts a form HTML control in the HTML portion of your ASP.NET Web pages. The <form> tag is inserted in an HTML control because it has the `runat="server"` attribute, as can be seen by clicking the HTML tab (see Figure 3.4).

**FIGURE 3.4**  
*The HTML portion of the ASP.NET Web page contains a form HTML control.*



The form HTML control is commonly referred to as a *Web form* or *server-side form*. The remainder of this book will refer to form HTML controls as either *Web forms* or *server-side forms*.

When creating an ASP.NET Web page that accepts user input—such as our financial calculator, which accepts three inputs from the user—it is required that the Web controls for user input (the TextBoxes, DropDownLists, CheckBoxes, and so on) be placed within a Web form. We will discuss why this needs to be done in Hour 9, “Web Form Basics.”

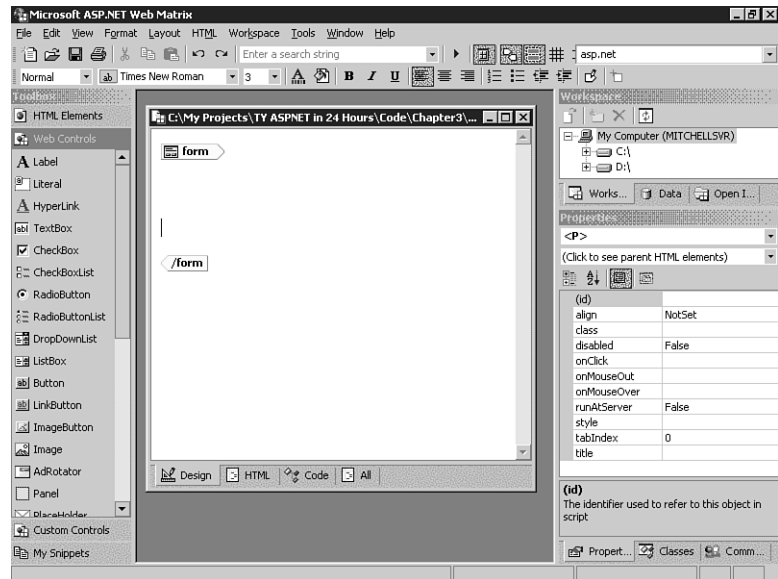


Because our user input Web controls need to be within the Web form, it is important that we turn on glyphs so that we can see the Web form’s start and finish. Then we can be sure that the controls we drop onto the designer from the Toolbox actually end up between the Web form’s start and end tags, as opposed to before or after the Web form.

## Adding the Three TextBox Web Controls

Let's start by adding the TextBox Web controls for our user's three inputs. First make sure that you are in the Design tab. Next place your mouse cursor between the Web form glyphs and click the left mouse button so that the designer receives focus and there is a flashing cursor between the Web form glyphs. Create some space between the Web form's start and end tags by hitting enter a few times. You should see something similar to Figure 3.5.

**FIGURE 3.5**  
*Hit Enter to create some space between the Web form start and end tags.*

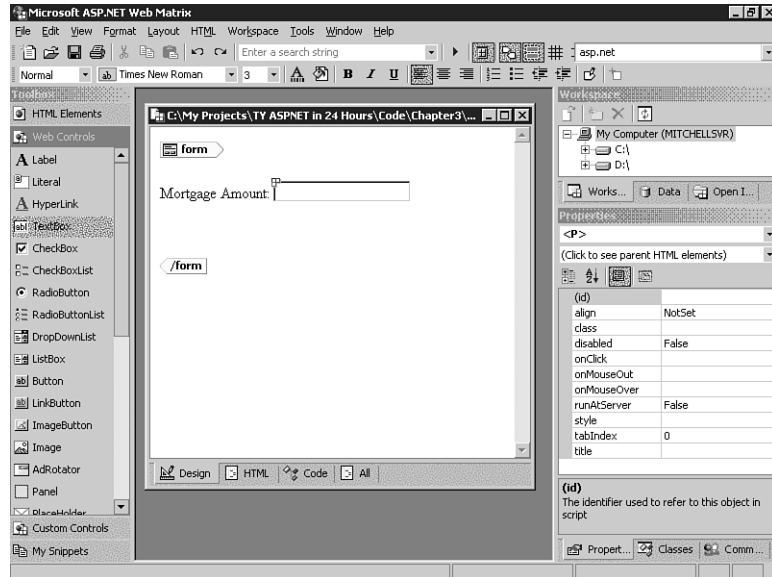


Before we drag a TextBox Web control to the designer, let's first create the title for the textbox we're going to add. Because the first input is the amount of the mortgage, start by typing in this title: **Mortgage Amount:.**

Next we want to add a TextBox Web control after this title. To accomplish this, make sure that the Web Controls tab from the Toolbox is selected, and then drag a TextBox control from the Toolbox and drop it into the designer after the "Mortgage Amount:" title. Take a moment to make sure your screen looks similar to the screenshot shown in Figure 3.6.

**FIGURE 3.6**

*At this point you should have a title and a single textbox, both inside the Web form tags.*



3



When dragging and dropping the TextBox Web control from the Toolbox, it is very important that the Web Control tab be selected so that you are indeed dragging and dropping TextBox Web controls. If the HTML Elements tab is selected on the Toolbox, you'll be placing standard HTML textboxes (textboxes without the `runat="server"` attribute present).

If you do not use TextBox Web controls, you will not be able to reference the values that the user entered in the TextBoxes in the source code portion of the ASP.NET Web page. Therefore, be certain that when dragging and dropping TextBoxes onto the designer for this exercise, you are dragging TextBox Web controls, not HTML textboxes.

Currently, the TextBox Web control we just added has its ID property set to `TextBox1`. Because we will later need to refer to this ID in order to determine the value of the beginning retirement balance entered by the user, let's choose an ID value that is representative of the data found within the TextBox. Specifically, change the ID property to `loanAmount`.



To change a Web control's ID property, click the Web control in the designer, which will load the Web control's properties in the Properties window in the lower right-hand corner. Scroll to the top of the Properties pane until you see the ID property. This is the property value that you should change. Note that in the list of properties in the Properties pane, the ID property is denoted as (ID).

Now let's add the second textbox, the mortgage's interest rate. Add it just as we did the previous TextBox Web control by first creating a title for the TextBox. Type in the title **Annual Interest Rate:**. Next drag and drop a TextBox Web control after this title and change the TextBox's ID property to rate.

Finally, add the third textbox, the duration of the mortgage. Start by adding the title **Mortgage Length:**, and then drag and drop a TextBox Web control after the title. Set this TextBox's ID to mortgageLength.



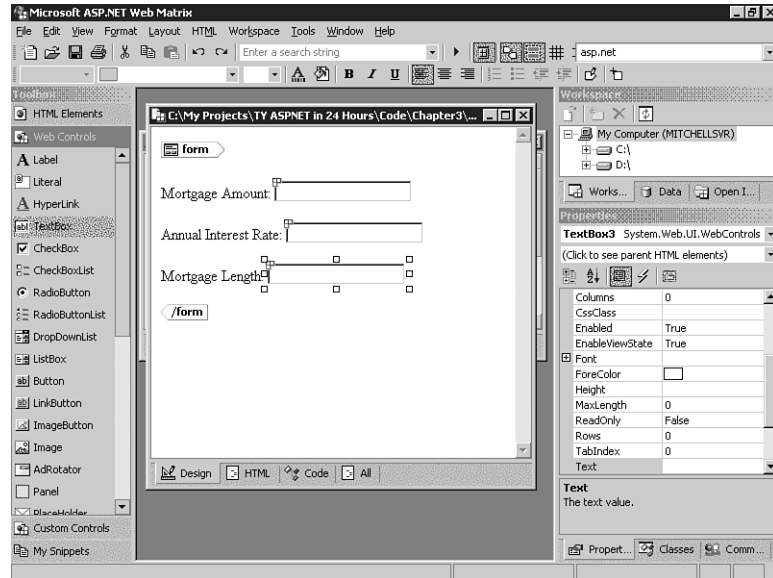
You might want to type in some text after each TextBox Web control to indicate the units that should be entered into the textbox. For example, after the "Annual Interest Rate" textbox, you might want to add a percent sign so that the user knows to enter this value as a percentage. Similarly, you might want to enter the word "years" after the "Mortgage Length" TextBox.

Figure 3.7 contains a screenshot of the Design tab after all three input TextBox Web controls have been added.



The screenshot in Figure 3.7 shows the TextBox Web control titles in the standard font. Feel free to change the font or the aesthetics of the HTML portion however you see fit. Just be sure to have three TextBox Web controls inside of the Web form.

**FIGURE 3.7**  
*A screenshot of the Design tab, shown after all three TextBox Web controls have been added.*



## Adding the Compute Monthly Cost Button

After the user has entered inputs into the three TextBox Web controls, we want to be able to take that information and perform our financial calculation. Realize, though, that when the ASP.NET engine executed the ASP.NET Web page, it converted the TextBox Web controls into HTML `<input>` tags.

As we'll discuss in much greater detail in the next hour, "Dissecting Our First ASP.NET Web Page," when the users visit the `FinancialCalculator.aspx` ASP.NET Web page via their browsers, they are receiving HTML that contains a `<form>` tag and, within it, three `<input>` textbox tags. This HTML markup, when rendered by a browser, displays three textboxes, as shown in Figure 3.7. In order for the calculation to take place, the inputs entered by the user must be submitted back to our ASP.NET Web page (`FinancialCalculator.aspx`). Once our ASP.NET Web page receives these user-entered values, it can perform the financial computation and return the results.

In order for an HTML form to be submitted, the user needs a button that, when clicked, causes the form to be submitted. We can add such a button by adding a Button Web control to our ASP.NET Web page.



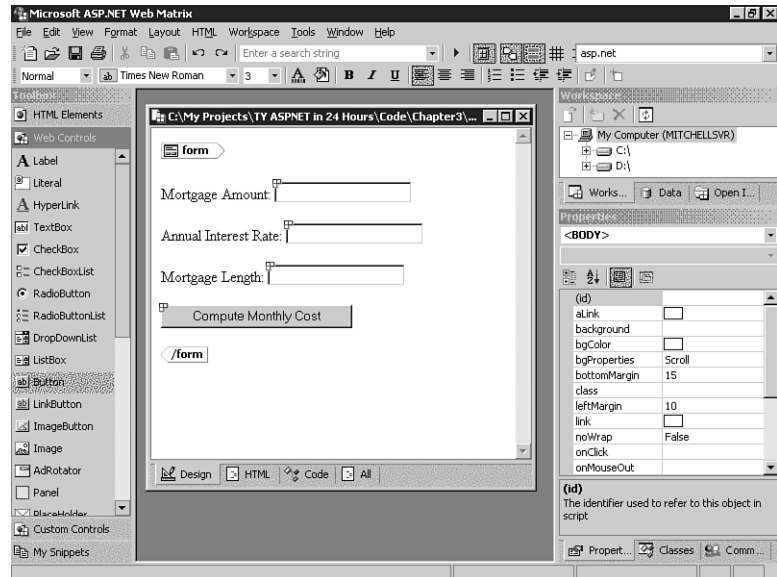
We will discuss the specifics involved with collecting and computing user input in Hour 9, “Web Form Basics.”

To add a Button Web control, first ensure that the Web Controls tab in the Toolbox is selected. Then drag the Button Web control from the Toolbox onto the designer, dropping it after the last input title and textbox.

When dropping a Button Web control onto the designer, the button’s caption reads “Button.” To change this, click the button and then in the Properties pane change the Text property from Button to Compute Monthly Cost. This will change the caption on your button to “Compute Monthly Cost.” Also, while in the Properties pane, change the button’s ID property—listed in the property pane as (ID)—from the default Button1 to performCalc.

Take a moment to make sure that your screen looks similar to the screenshot in Figure 3.8.

**FIGURE 3.8**  
*A Button Web control has been added.*



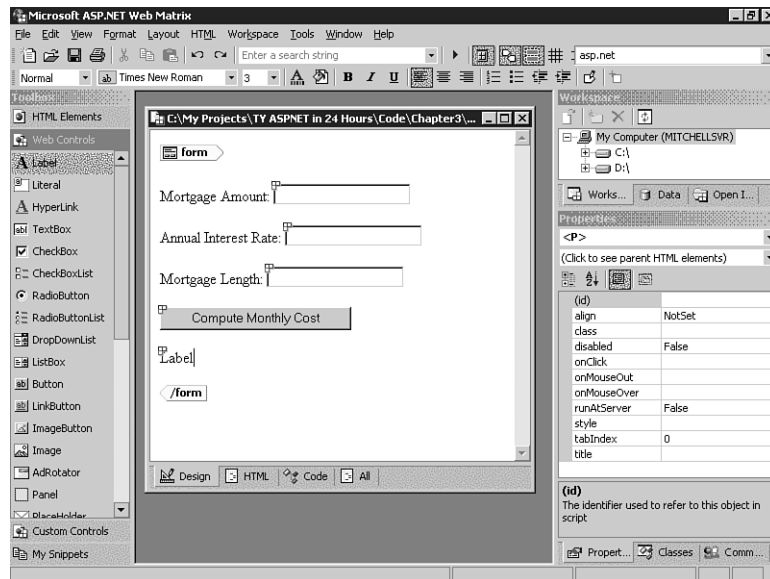
## Creating a Label Web Control for the Output

The final piece we need to add to our user interface is a Label Web control that will display the output of our financial calculation. Because the Label Web control will display the output (the amount of money the mortgage costs per month), the Web page's final result will appear wherever you place the Web control. Therefore, if you want the output to appear at the bottom of your ASP.NET Web page, simply drag and drop a Label Web control after the existing content in the designer. If you want the output to appear at the top of the Web page, place it before the existing content in the designer.

To add the Label Web control, drag and drop it from the Toolbox and onto the designer. Once you have added the Label Web control, you will see that it displays the message "Label." The Label Web control displays the value of its Text property, which is configurable via the Properties pane. Figure 3.9 is a screenshot of the designer with the Label Web control added.

**FIGURE 3.9**

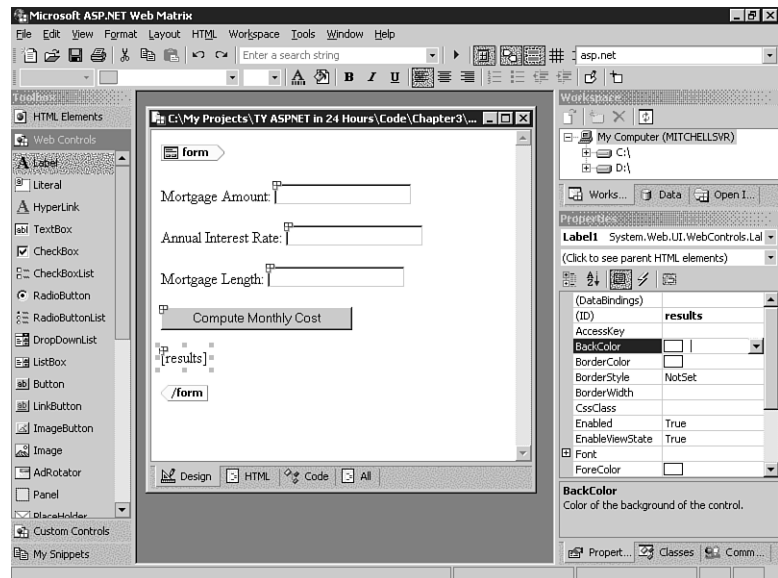
*A Label Web control has been added to the ASP.NET Web page.*



Because we don't want this label to display any content until the user has entered their three inputs and the calculation has been performed, clear out the Label's Text property. To clear out a property value for the Label Web control, first click the Label Web control so that its properties are loaded in the Properties pane. Then, in the Properties pane, locate the Text property and erase the Text property value by clicking the Text property's value and hitting backspace until all of the characters have been erased.

Once you clear out the Label's Text property, the designer will show the Label Web control as its ID property, enclosed by brackets. Currently, the Label Web control's ID property is `Label1`, meaning that in the designer you should see the Label Web control displayed as: `[Label1]`. Go ahead and change the ID property of the Label Web control from `Label1` to `results`, which should change the label's display in the designer from `[Label1]` to `[results]`. Figure 3.10 shows a screenshot of the designer after the Label Web control's property has been changed to `results`.

**FIGURE 3.10**  
*The Label Web control's ID has been changed to results.*



## Completing the User Interface

At this point we have added the vital pieces of the user interface. This was accomplished using the Web Matrix Project's WYSIWYG editor in a fraction of the time it would have taken to enter the HTML markup and Web control syntax manually.



To fully appreciate the amount of HTML markup the Web Matrix Project generated for us automatically, click the HTML tab.

If you want to add additional user interface elements at this time, perhaps a bold, centered title at the top of the Web page or a brief set of instructions for the user, feel free to do so.

Now that we have created the HTML portion of the ASP.NET Web page, we are ready to create the source code portion in the next section.



HTML markup and Web controls not used for user input may appear either within the Web form tags or outside of these tags. The only things that *must* be placed within the Web form tags are the Web controls that collect user input (the TextBoxes and Button).

## Writing the Source Code for the ASP.NET Web Page

3

Now that we have completed the HTML portion of our ASP.NET Web page, all that remains is the source code. The source code will read the user's inputs and perform the necessary calculations to arrive at the monthly cost for the mortgage.

In the previous hour we looked at the `Page_Load` event handler. This event handler, which you can include in your ASP.NET Web page's source code portion, is executed each time the Web page is loaded. We will not be placing the source code to perform the monthly mortgage cost calculation in this event handler, though, because we do not want to run the calculation until the user has entered the loan amount, interest rate, and duration, and has clicked the Compute Monthly Cost button.

Button Web controls have a `Click` event, which fires when the button is clicked. Therefore, what we want to do is write our own event handler and associate it with the Compute Monthly Cost button's `Click` event. This way, whenever the Compute Monthly Cost button is clicked, the event handler that we provide will be executed. All that remains, then, is to place the source code that performs the computation inside this event handler.

Adding event handlers to a Button Web control's `Click` event is quite easy to accomplish with the Web Matrix Project. From the designer, simply double-click the Button whose `Click` event you would like to provide an event handler for. Once you double-click the Button Web control, you will be whisked to the Code tab, where you should see the following source code already entered:

```
Sub performCalc_Click(sender As Object, e As EventArgs)
```

```
End Sub
```

These two lines of code are the shell for the button's `Click` event handler. Note that the event handler is named `performCalc_Click`. More generally, it is named `buttonID_Click`, where `buttonID` is the value of the button's ID property. (Recall that after adding the Button Web control, we changed its ID from `Button1` to `performCalc`.)

Any code that you write between these two lines will be executed whenever the `performCalc` button is clicked. Because we want to compute the monthly cost of the mortgage when the `performCalc` button is clicked, the code to perform this calculation will appear within the `performCalc_Click` event handler.

## Reading the Values in the TextBox Web Controls

In order to calculate the monthly cost of the mortgage, we must first be able to determine the values entered by the user into the three `TextBox` Web controls. Before we look at the code to accomplish this, let's take a step back and reexamine Web controls, a topic we touched upon lightly in the previous hour.

Recall from Hour 2's discussion that when the ASP.NET engine is executing an ASP.NET Web page, Web controls are handled quite differently from standard HTML elements. Standard HTML markup is passed directly from the ASP.NET engine to the Web server without any translation; with Web controls, however, an object is created that represents the Web control. The object is created from the class that corresponds to the specific Web control. That is, a `TextBox` Web control has an object *instantiated* from the `TextBox` class, whereas a `Label` Web control has an object *instantiated* from the `Label` class.



Recall that a class is an abstract blueprint, whereas an object is a concrete instance. When an object is created, it is said to have been *instantiated*. The act of creating an object is often referred to as *instantiation*.

Each of these classes has various properties that describe the state of the Web control. For example, the `TextBox` class has a `Size` property that indicates how many columns the textbox has. Both the `TextBox` and the `Label` classes have `Text` properties that indicate that Web control's text content.



The classes that represent various Web controls are classes in the .NET Framework. We will discuss how to find the properties, methods, and events for these classes in future hours.

The primary benefit of Web controls is that their properties can be accessed in the ASP.NET Web page's source code section. Because the `Text` property of the `TextBox` Web control contains the content of the textbox, we can reference this property in the Compute Monthly Cost button's `Click` event handler to determine the value entered by the user into each textbox.

For example, to determine the value entered into the Mortgage Amount textbox, we could use the following line of code:

```
loanAmount.Text
```

When the ASP.NET engine creates an object for the Web control, it names the object the value of the Web control's ID property. Because `loanAmount` is the ID of the Mortgage Amount `TextBox` Web control, the object created representing this Web control is named `loanAmount`. To retrieve the `Text` property of the `loanAmount` object, we use the syntax `loanAmount.Text`.



Don't worry if the syntax for retrieving an object's property confuses you. We will be discussing the syntax and semantics of Visual Basic .NET in greater detail in Hour 5, "Visual Basic .NET Variables and Operators."

3

## The Complete Source Code

Listing 3.1 contains the complete source code for our ASP.NET Web page. Take a moment to enter the source code shown below into the `performCalc` Button's `Click` event handler. (You should do this from the `Code` or `All` tab.)



Keep in mind that the line numbers shown in Listing 3.1 should *not* be typed in as well. The line numbers are present in the code listing only to help reference specific lines of the listing when discussing the code.

### LISTING 3.1 The Computation Is Performed in the `performCalc` Button's `Click` Event Handler

```
1: Sub performCalc_Click(sender As Object, e As EventArgs)
2:     'Specify constant values
3:     Const INTEREST_CALCPS_PER_YEAR as Integer = 12
4:     Const PAYMENTS_PER_YEAR as Integer = 12
5:
6:     'Create variables to hold the values entered by the user
7:     Dim P as Double = loanAmount.Text
```

*continues*

**LISTING 3.1** Continued

```
8: Dim r as Double = rate.Text / 100
9: Dim t as Double = mortgageLength.Text
10:
11: Dim ratePerPeriod as Double
12: ratePerPeriod = r/INTEREST_CALCS_PER_YEAR
13:
14: Dim payPeriods as Integer
15: payPeriods = t * PAYMENTS_PER_YEAR
16:
17: Dim annualRate as Double
18: annualRate = Math.Exp(INTEREST_CALCS_PER_YEAR * Math.Log(1+ratePerPeriod)) - 1
19:
20: Dim intPerPayment as Double
21: intPerPayment = (Math.Exp(Math.Log(annualRate+1)/payPeriods) - 1) *
    payPeriods
22:
23: 'Now, compute the total cost of the loan
24: Dim intPerMonth as Double = intPerPayment / PAYMENTS_PER_YEAR
25:
26: Dim costPerMonth as Double
27: costPerMonth = P * intPerMonth/(1-Math.Pow(intPerMonth+1, -payPeriods))
28:
29:
30: 'Now, display the results in the results Label Web control
31: results.Text = "Your mortgage payment per month is $" & costPerMonth
32: End Sub
```

An in-depth discussion of the code in Listing 3.1 will have to wait until the next hour. For now, simply enter the code as is, even if there are parts of it you don't understand. One thing to pay attention to, though, is in lines 7 through 9. In these three lines we are reading the values of the three `TextBox` Web controls and assigning the values to variables.



If the source code in Listing 3.1 has you hopelessly lost and confused, don't worry. The point of this hour is to get you creating a useful ASP.NET Web page quickly; we will take the time needed to dissect the HTML and source code portions of this ASP.NET Web page in the next two hours.



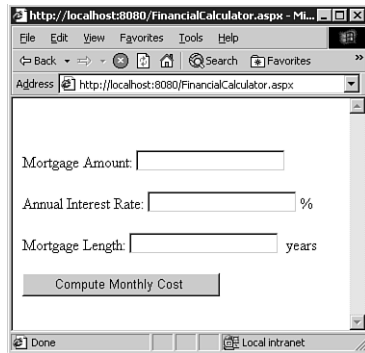
The mathematical equations used to calculate the monthly interest cost can be found at <http://www.faqs.org/faqs/sci-math-faq/compoundInterest/>. A more in-depth discussion of these formulas can be found at [http://people.hofstra.edu/faculty/Stefan\\_Waner/RealWorld/Summary10.html](http://people.hofstra.edu/faculty/Stefan_Waner/RealWorld/Summary10.html).

## Testing the Financial Calculator

Now that we have completed the HTML and source code portions of our ASP.NET Web page, it's time to test. First make sure that you have saved the ASP.NET Web page since entering the source code in Listing 3.1. Next view the ASP.NET Web page through your browser. When first visiting the page, you should see three empty textboxes and the Compute Monthly Cost button, as shown in Figure 3.11.

**FIGURE 3.11**

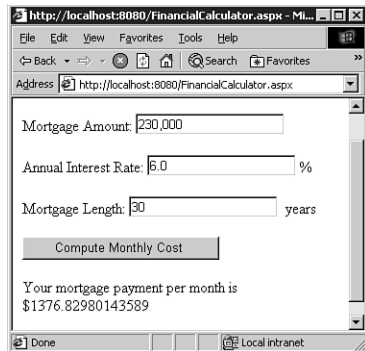
*When the ASP.NET page is first visited, three textboxes await user input.*



Now go ahead and enter some values into the textboxes and then click the Compute Monthly Cost button. When this button is clicked, the monthly cost is displayed beneath the textboxes and button, as shown in Figure 3.12.

**FIGURE 3.12**

*The output of the financial calculator is displayed when the Button is clicked.*

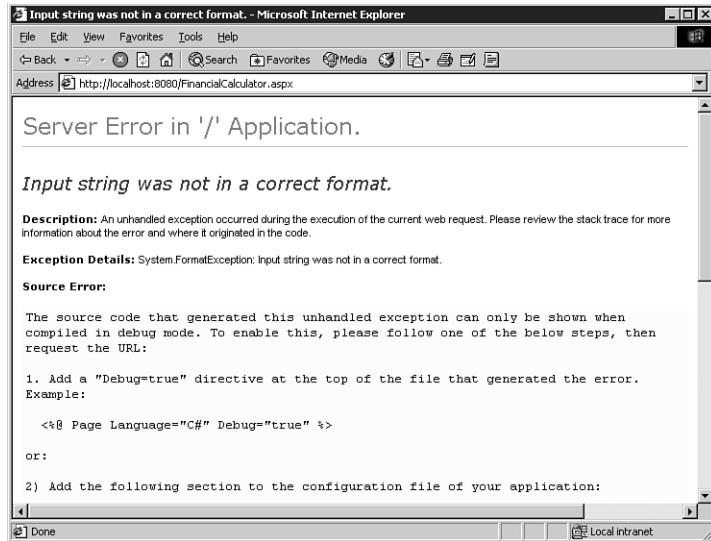


## Testing Erroneous Input

Part of testing is not only testing expected inputs but also testing unexpected ones. For example, what will happen if the user enters into the Mortgage Length textbox a value of “Jisun”? Obviously, this is not a valid number of years. Entering such an erroneous value will cause a run-time error, as shown in Figure 3.13.

**FIGURE 3.13**

*A run-time error will occur if the input is not in proper format.*



Errors such as those shown in Figure 3.13 are an eyesore. Rather than displaying such error messages when the user enters erroneous input, it would be better to display a simple error message next to the textbox(es) with erroneous input, explaining that the input is not in the right form.

The process of ensuring that user input is in the correct format is known as *input validation*. Fortunately, input validation is incredibly easy with ASP.NET. We’ll examine ASP.NET’s input validation features in Hour 12, “Validating User Input with Validation Controls.”



Earlier in this hour we discussed the importance of planning the user interface and functionality of an ASP.NET Web page prior to creating the page. Not only is it important to plan how the ASP.NET Web pages should work, but it is also important to plan on how the ASP.NET Web page should behave when things don’t necessarily go according to plan.

## Summary

In this hour we saw how to create our first useful ASP.NET Web page. We started by outlining the features we wanted to include in our ASP.NET Web page, including the output and needed inputs. We then briefly discussed what the user interface should look like.

Next we implemented the user interface by completing the HTML portion of the ASP.NET Web page. Using the Web Matrix Project's WYSIWYG editor, it was simply a matter of typing in the textbox labels and dragging and dropping the needed TextBox, Button, and Label Web controls.

After the HTML portion, the source code portion was entered. The code to perform the calculation was inserted in an event handler for the Compute Monthly Cost button's `Click` event. This had the effect of having the entered code executed whenever the user clicked the Compute Monthly Cost button.

Finally, we tested the ASP.NET Web page by visiting it with a Web browser and entering some values for the three textboxes.

In this hour we did not spend much time discussing the source code or the specifics of the Button Web control's `Click` event and corresponding event handler. We will touch upon these issues in detail in the next two hours.

3

## Q&A

**Q Can I use HTML controls instead of Web controls for the textboxes in the HTML portion of the ASP.NET Web page?**

**A** Yes, but I would advise against it, in large part because the Web Matrix Project's Toolbox does not contain support for dragging and dropping HTML controls. Rather, you would have to enter the markup for the HTML controls by hand in the HTML tab. That is, there are no HTML controls in the Web Matrix Project's Toolbox, just Web controls and HTML elements.

Additionally, we will be using Web controls extensively throughout this book, so I would encourage you to familiarize yourself with adding Web controls to an ASP.NET Web page and consider using HTML controls only when an example explicitly mentions their use.

**Q How do I associate "event code" with a Web control that I've placed on a Web Form?**

**A** In this hour we saw how to have the Button Web control's `Click` event associated with an event handler provided in the source code section. We accomplished this

by simply double-clicking the Button Web control in the designer. Realize that, behind the scenes, the Web Matrix Project is performing a number of steps when you double-click the Button. Each Web control has a *default event*. When the Web control is double-clicked in the designer, an event handler is created for this default event. (Note that the Button Web control's default event is the `Click` event.)

Adding an event handler for an event other than a Web control's default event involves a more thorough discussion than we are ready for at this point. We'll examine adding event handlers for events other than the Web control's default event in the next hour.

**Q What would happen if I placed the financial calculation code in the `Page_Load` event instead of the button `Click` event handler?**

**A** Recall that the source code in the `Page_Load` event handler executes every time the ASP.NET Web page is requested. When the page is visited for the first time by the user, the user has yet to enter the loan principal, interest rate, and duration. Therefore, in attempting to compute the calculation, we will get an error.

Because we want to perform the calculation only *after* the user has provided the required inputs, the source code for the calculation is placed in the button's `Click` event handler.

## Workshop

### Quiz

1. Why is the design requirements phase of software development an important one?
2. How can one add a TextBox Web control to an ASP.NET Web page using the Web Matrix Project?
3. Why did we add a Label Web control to our ASP.NET Web page's HTML portion?
4. What will the ASP.NET Web page's output be if the user enters invalid characters into the textboxes—for example, if under the Mortgage Amount textbox, the user enters "Scott"?
5. How do you add an event handler for a Button Web control's `Click` event with the Web Matrix Project?
6. When using a TextBox Web control, what property is referenced to determine the value entered by the user?

## Answers

1. The design requirements phase outlines the specific features for the software project and also outlines the user interface. It is an important stage because by enumerating the features, you—and your boss and client—can easily determine the current progress of the project. Furthermore, there is no ambiguity as to what features should and should not be included.
2. To add a TextBox Web control, simply click the TextBox Web control from the Toolbox and drag it onto the designer.
3. A Label Web control was added to the ASP.NET Web page's HTML portion to indicate where the output of the financial calculator would appear. Without using a label, we could output the results using only `Response.Write()` statements. Recall from the previous hour that this would have the effect of omitting the results before the HTML portion of the ASP.NET Web page. By using a Label Web control, then, we have more flexibility over where the output appears in the HTML portion.
4. If the user provides invalid input, a run-time error will occur.
5. To add an event handler for a Button Web control's `Click` event, simply double-click the button that you wish to add an event handler for.
6. The `Text` property contains the value entered by the user. To reference this property in an ASP.NET Web page's source code portion, we can use

```
TextBoxID.Text
```

## Exercises

1. In this hour we saw how to use the Web Matrix Project to create an ASP.NET Web page with TextBox Web controls, a Button Web control, and a Label Web control. Using this knowledge, let's create an ASP.NET Web page that will prompt the user for his name and age. Once the user provides this information and clicks the submit button, the ASP.NET Web page will display a message whose content depends on the user's age.

This ASP.NET Web page will need to have two TextBox Web controls, a Button Web control, and a Label Web control. Set the TextBox Web controls' ID properties to name and age. The Button Web control should have its `Text` property set to `Click Me`. Set the Label Web control's ID property to `results` and clear out its `Text` property. You will then need to create an event handler for the Button Web control's `Click` event—recall that this is accomplished by simply double-clicking the Button in the designer.

Now, in the `Click` event handler, you need to determine what message to display, based on the user's age. The code for this will look like

```
If age.Text < 21 then
    results.Text = name.Text & ", you are a youngster!"
End If
```

```
If age.Text >= 21 AND age.Text < 40 then
    results.Text = name.Text & ", you are an adult."
End If
```

```
If age.Text >= 40 then
    results.Text = name.Text & ", you are over the hill!"
End If
```

Once you have entered the preceding source code into the `Button` Web control's `Click` event handler, save the ASP.NET Web page and test it by visiting it through a browser.

2. For more practice with the Web Matrix Project, take a moment to enhance the user interface of the `FinancialCalculator.aspx` Web page we created in this hour. A couple suggested enhancements, of many possible, include displaying the `TextBox` Web control titles in a more appealing font and adding some text at the top of the Web page explaining the purpose of the financial calculator.