THE EXPERT'S VOICE® IN .NET

FIFTH EDITION

Pro ASP.NET 4.5 in VB

DISCOVER THE POWER OF MICROSOFT'S MOST SOPHISTICATED WEB TECHNOLOGY YET

Dan Mabbutt, Adam Freeman, and Matthew MacDonald

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Contents at a Glance

Introduction

The book you're reading is a comprehensive guide to programming in Microsoft's software development technology for the web, ASP.NET. Microsoft supports two languages "across the board" for their software development products—C# and Visual Basic .NET. (The alternative .NET development technology, Mono, also supports both C# and VB.) Apress also provides the same two languages for this comprehensive guide. The examples in this book are all written in VB.NET, but if you're a C# programmer, you can find basically the same book written for C#.NET.

The very existence of parallel books using either C# or Visual Basic that explore nearly everything of importance in all ASP.NET using the latest Microsoft versions is clear evidence that Microsoft is doing a great job of making Visual Basic and C# equivalent in technical terms. After source code is compiled, the resulting CLI (Common Language Interface) code really is equivalent. The only reason that a programmer—or an organization—would choose one language over another is individual choices about what people are familiar with and can work with more efficiently. For me, that's Visual Basic. For my co-author Adam, that's C#. High-end programmers who may have started with C++, Java, or C often prefer C# because that's the way they're used to seeing code. But there are a lot of us who don't too. For example, trainer Andy Brown has written an entertaining argument, "10 Reasons Why Visual Basic is Better Than C#". In the comments section of that article, the arguments go on for pages. Our view is that you should use what works for you. (Or, what your employer has decided will work.)

We start in Part 1 by establishing a foundation of tools and understanding that you use throughout the rest of the book. Part 1 might be considered as a "book in a book" because it can stand on its own in explaining what you need to write basic systems using ASP.NET. In Chapter 1, a simple ASP.NET system is developed that accepts user input, checks it against previous input, and saves it in a data store–in addition to showing you how to install Visual Studio Express 2012 for Web. By the end of Part 1, the basics of a retail store application has been developed.

Part 2 builds on Part 1 by explaining more advanced features of the ASP.NET platform with a special emphasis on those features that work directly with the HTTP requests.

The focus of Part 3 is web Forms, and those endlessly useful objects, Controls. Controls are the essential component of nearly all real-world applications.

In Part 4, we switch our focus from the server side of development to the client side with discussions of scripts and style sheets, web services, and model binding.

Although this book is intended to be comprehensive and show how to use the most advanced features of ASP.NET, you don't have to start out as an advanced programmer to get the best from it. We assume that you know how to program in Visual Basic .NET and that you have a reasonable understanding of how the web works and work from there. All the examples are available for downloading at the Apress site. The illustrations are generated directly from the code that you can download. The development environment itself, Visual Studio Express 2012 for Web, is free and downloadable at Microsoft's site. Although the examples were developed and tested in a Windows 8 environment, supported previous versions of Windows—and especially Windows 7—should work just as well for you.

PART 1

Getting Started

We start this book by jumping straight into ASP.NET and creating a simple application. We'll then explain the Visual Basic language features and development tools that are needed for ASP.NET development and use them to create a realistic web application called SportsStore.

CHAPTER 1

Your First ASP.NET Application

The best way to get started with ASP.NET is to jump right in. In this chapter, we will show you how to get set up for ASP.NET development and build your first ASP.NET application. The application we will build is simple, but it allows us to show you how to prepare your workstation for ASP.NET development, how the ASP.NET development tools work and—most importantly—how quickly you can get up and running with ASP.NET. We'll provide some context and background about the ASP.NET Framework in the next chapter, but this book focuses on coding so that's what we are going to start with.

Preparing Your Workstation

You only need two things for ASP.NET development—a Windows 7 or Windows 8 workstation and Visual Studio, which is the Microsoft development environment. You probably have a Windows installation already, but you can usually find some pretty good deals if you need to buy a copy. Microsoft has discount schemes you can use if you are a student or teacher, or if you want to upgrade schemes from older Windows versions. Microsoft also has subscription based products if you want wider access to their software products such as their MSDN subscriptions. You can get a 90-day trial of Windows 8 from `msdn.microsoft.com/en-us/windows/apps` if you don't have Windows and you would like to try out ASP.NET development on Windows 8 without making a commitment.

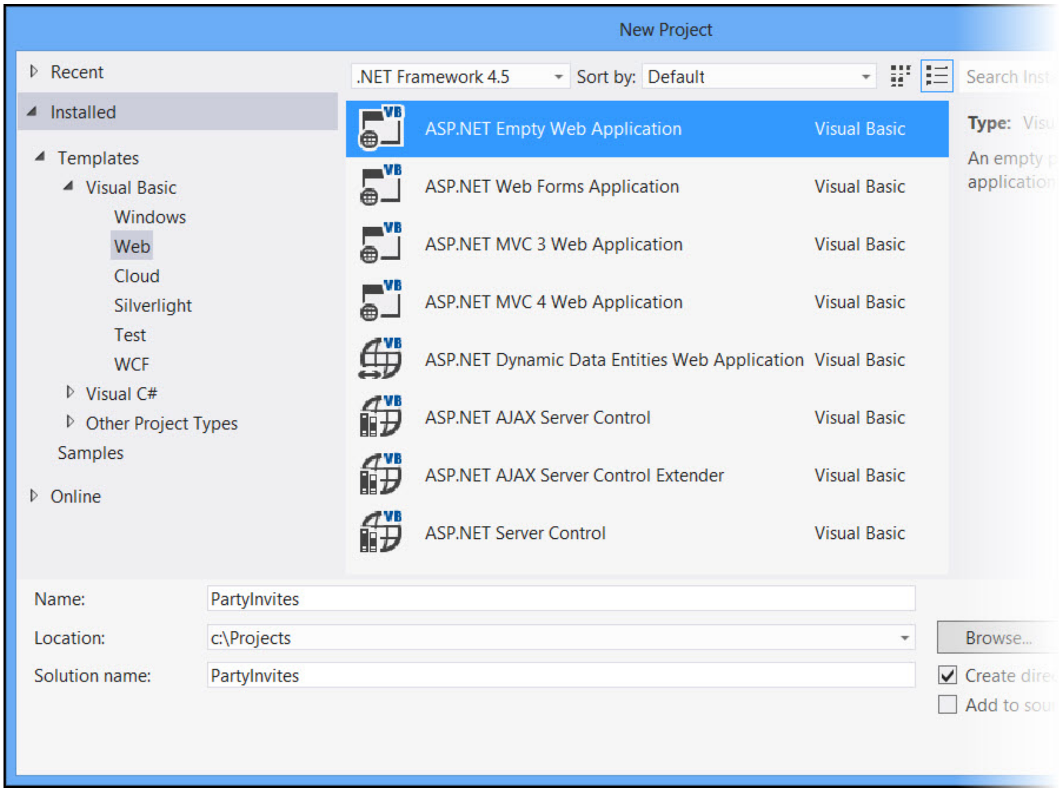
You need Visual Studio 2012 to build applications with ASP.NET 4.5, the version of the ASP.NET Framework we use in this book. Several different editions of Visual Studio 2012 are available, but we will be using the one that Microsoft offers free of charge, *Visual Studio Express 2012 for Web*. Microsoft adds some nice features to the paid-for editions of Visual Studio, but you won't need them for this book. All figures throughout this book have been taken using the Express edition running on Windows 8. You can download the Express edition from `www.microsoft.com/visualstudio/eng/products/visual-studio-express-products`. There are several different editions of Visual Studio 2012 Express, each of which is used for a different kind of development—make sure that you get the Web edition, which supports ASP.NET applications.

Tip You can use any edition of Visual Studio 2012 for the examples in this book. You will see slight differences in some of the dialog windows and the menu and toolbar configurations, but otherwise you will be just fine.

Creating a New ASP.NET Project

Start Visual Studio 2012 and select `New Project` from the `File` menu. You will see the `New Project` dialog window which—as the name suggests—you use to create new Visual Studio projects.

You will see a list of the available projects types in the left-hand panel of the dialog window. Navigate to `Installed` > `Templates` > `Visual Basic` > `Web` and you will see the set of ASP.NET projects available, as shown in Figure 1-1.



ASP.NET Empty Web Application	Visual Basic
ASP.NET Web Forms Application	Visual Basic
ASP.NET MVC 3 Web Application	Visual Basic
ASP.NET MVC 4 Web Application	Visual Basic
ASP.NET Dynamic Data Entities Web Application	Visual Basic
ASP.NET AJAX Server Control	Visual Basic
ASP.NET AJAX Server Control Extender	Visual Basic
ASP.NET Server Control	Visual Basic

Type: Visu
An empty
applicatio

Name: PartyInvites
Location: c:\Projects
Solution name: PartyInvites

***Figure 1-1.** The New Project dialog window*

■ **Tip** Make sure you select `Visual Basic` and not `Visual C#`. You'll get some very odd behavior and errors if you try to follow our Visual Basic examples in a C# project.

Select the `ASP.NET Empty Web Application` item from the central panel of the dialog—some of the names of the different project types are similar, so make sure that you get the right one. Make sure that `.Net Framework 4.5` is selected in the drop-down menu at the top of the screen and set the Name field to `PartyInvites`. Click the OK button to create the new project.

■ **Tip** Visual Studio sets the `Solution Name` field to `PartyInvites` to match the project name. A Visual Studio solution is a container for one or more projects, but most of the examples in this book will contain one project, which is typical for ASP.NET Framework development.

The `ASP.NET Empty Web Application` is the simplest of the project templates and creates a project that only contains a `Web.config` file that contains the configuration information for your ASP.NET application. Visual Studio shows you files in the Solution Explorer window, which you can see in Figure 1-2. Solution Explorer is the principal tool for navigating around your project.

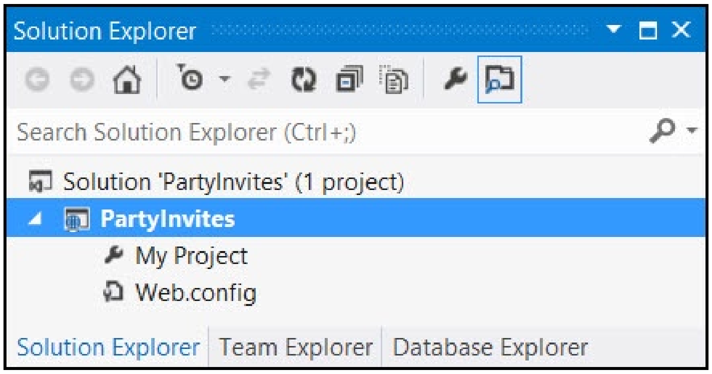


Figure 1-2. *The Visual Studio Solution Explorer window*

Adding a New Web Form

As you saw when you created the Visual Studio project, there are different kinds of ASP.NET applications. For the type of application we describe in this book, content is generated from a *Web Form*. This is a misleading name, as we explain Chapter 2, but for the moment it is enough to know that we add content to our application by adding new Web Form items.

To add a new Web Form to the project, right-click the `PartyInvites` project entry in the Solution Explorer window and select `Add` > `Web Form` from the pop-up menu. When prompted, enter `Default` as the name for the new item, as shown in Figure 1-3.

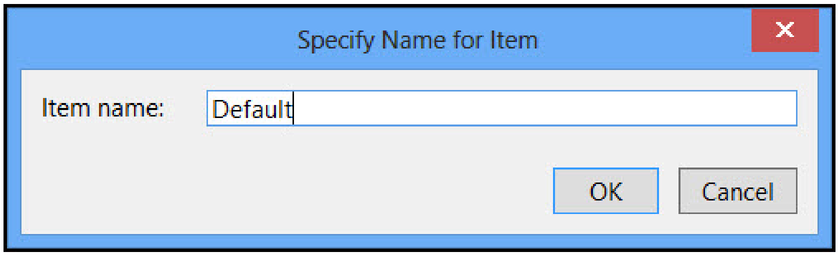


Figure 1-3. *Setting the name for the new Web Form*

■ **Note** Throughout this book, we build up each example so that you can follow along in your own Visual Studio project. If you don't want to follow along, you can download a complete set of example projects from `apress.com`. We have organized the examples by chapter and have included all the files you will need.

Click the OK button to dismiss the dialog and create the new item. You will see that Visual Studio has added a Default.aspx file to the project in the Solution Explorer and opened the file for editing. You can see the initial contents of the file in Listing 1-1.

Listing 1-1. The Initial Contents of the Default.aspx File

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="Default.aspx.vb"
Inherits="PartyInvites._Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

A Web Form file is, at its heart, an enhanced HTML file. The element that has the <% and %> tags gives away the fact this isn't a regular HTML file, as do the runat attributes in the head and form elements. We'll explain what all this means later, but for now we just want to emphasize that we really are working with HTML. In Listing 1-2, you can see that we have added some standard HTML elements to the Default.aspx file.

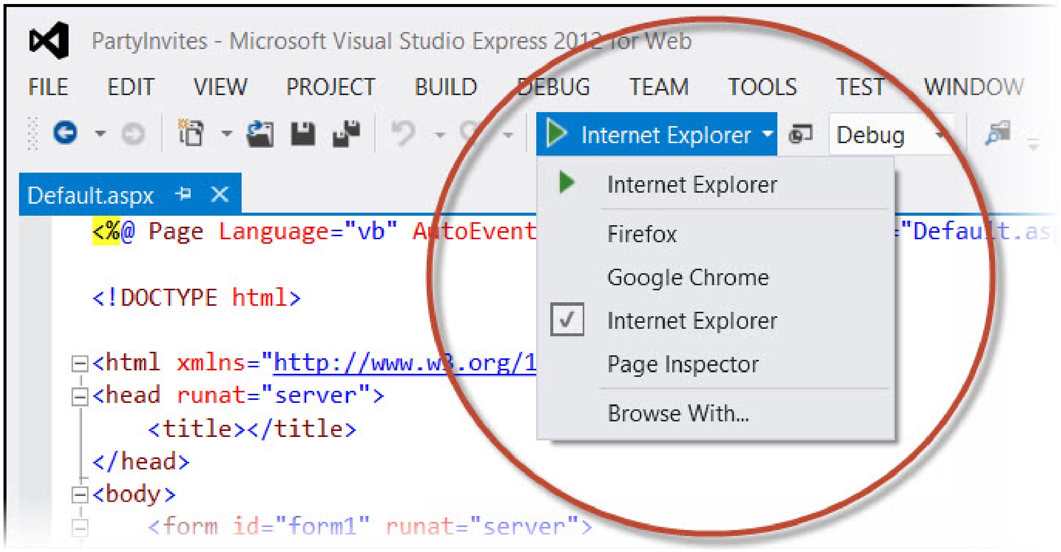
Listing 1-2. Adding Standard HTML Elements to the Default.aspx File

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="Default.aspx.vb"
Inherits="PartyInvites._Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>Hello</h1>
        <p>This is a new web form</p>
    </div>
    </form>
</body>
</html>
```

We have added an h1 and a p element containing some simple text. Nothing is specific to ASP.NET in these elements—they are standard HTML.

Testing the Example Application

The Visual Studio toolbar contains a drop-down list with the names of the browsers installed. You can see our list in Figure 1-4, which shows that we have several browsers installed. At the very least, you will have entries for `Internet Explorer` and `Page Inspector` (a tool that helps you debug your HTML and that we demonstrate later in Chapter 5).



```
<%@ Page Language="vb" AutoEvent                    ="Default.as

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
```

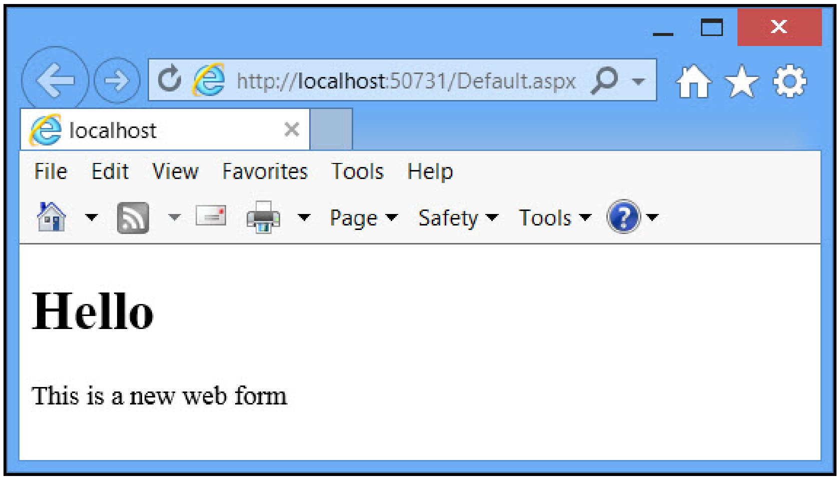
Figure 1-4. *Selecting a browser in Visual Studio*

We will be using Internet Explorer in this book, because it is always available on Windows workstations. There are occasions when we will use another browser to demonstrate a particular feature, but we'll always make it clear when this happens (and we'll show you the effect with a screenshot if you don't want to install additional browsers).

TESTING WITH MULTIPLE BROWSERS

Although we use Internet Explorer in this book, we recommend that you test your ASP.NET applications using as many browsers as possible, even if you don't want to install them on your development workstation. Browsers have reached rough parity with version 4 of the HTML and version 2 of the CSS standards, but we are now transitioning to HTML5 and CSS3. This means that there are some useful and exciting features available for web applications, but you have to test thoroughly to make sure that these new features are handled consistently across browsers.

Ensure that Internet Explorer is selected and then click the button or select `Start Debugging` from the Visual Studio Debug menu. Visual Studio compiles your project and opens a new browser window to display the Web Form, as shown in Figure 1-5. There isn't much content in the Web Form at the moment, but at least we know that everything is working the way that it should be.


http://localhost:50731/Default.aspx
localhost
File Edit View Favorites Tools Help
Page Safety Tools

Hello

This is a new web form

Figure 1-5. Displaying the Web Form in the browser

Here is the URL that Internet Explorer used for our example:

```
http://localhost:50731/Default.aspx
```

You will see a similar URL when you start the application, but it won't be exactly the same. You will see the `http://` part (specifying that the HTTP protocol is used) and the `localhost` part (this is a special name that refers to the workstation). The port part of this URL, `50731` in our case, is assigned randomly and you will see a different number. The last part of the URL, `Default.aspx`, specifies that we want the contents of our `Default.aspx` file. The result of processing is file is what you can see in the browser window.

So what does this URL relate to? Visual Studio 2012 includes *IIS Express*, which is a cut-down development version of the Microsoft application server used to run ASP.NET applications. IIS Express is installed automatically and you may see an icon in the notification window when it is running. (It's turned off by default.) You can right-click the `Customize` icon to see a list of the ASP.NET applications that you have running and open a browser window to view them, as shown in Figure 1-6.

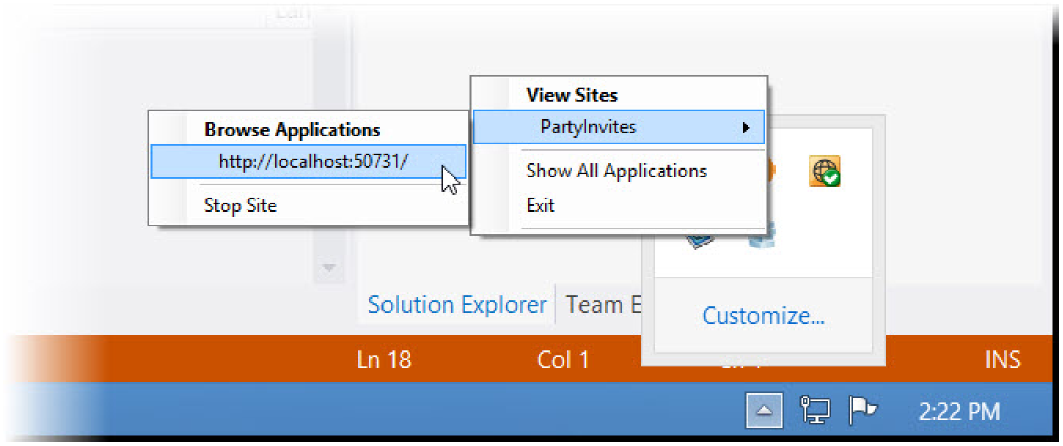


Figure 1-6. Interacting with IIS Express

When you used Visual Studio to run the application, IIS Express was started and it began listening for requests (on port 50731 for us, and most likely a different port for you). Once IIS Express had started up, Visual Studio created a new Internet Explorer window and used it to navigate to the URL which loads our `Default.aspx` file from IIS Express.

You can see the HTML that IIS Express and the ASP.NET Framework (which is integrated into IIS) sent to the browser by right-clicking in the browser window and selecting `View Source`. We have shown the HTML in Listing 1-3 and you will notice that it is different from the contents of the `Default.aspx` file.

Listing 1-3. The HTML Sent to the Browser by IIS Express in Response to a Request for Default.aspx

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
</title></head>
<body>
    <form method="post" action="./" id="form1">
<div class="aspNetHidden">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="oyo4u1kkzj6S7BwWSyQhmNtXxYEurT3+9lQDFO1jaP+rF+dFysFCL3FkuF66MzV
+ByArIcqgvdyJOZhXAwYEaiZtgGeakEPfZXMlj7q8UeQ=" />
</div>
    <div>
        <h1>Hello</h1>
        <p>This is a new web form</p>
    </div>
    </form>
</body>
</html>
```

The HTML sent to the browser is the result of the ASP.NET framework processing our `Default.aspx` file. The `<%` and `%>` tags have been removed and a hidden `input` element has been added, but because our `Default.aspx` file doesn't do anything interesting at the moment, the file contents are passed to the browser largely unmodified.

It may not seem like it, but you have created a very simple ASP.NET web application. These are the key points to bear in mind:

1. The user requests URLs that target Web Form files we add to the project.
2. The requests are received by IIS Express, which locates the request file.
3. IIS Express processes the Web Form file to generate a page of standard HTML.
4. The HTML is returned to the browser where it is displayed to the user.

This is the essence of any ASP.NET application. Our goal is to take advantage of the way that the ASP.NET Framework processes Web Forms files to create more complex HTML and more complex sequences of user interactions. In the sections that follow, we'll build on this basic foundation.

Creating a Simple Application

In the rest of this chapter, we will explore some of the basic ASP.NET features used to create a simple data-entry application. We will pick up the pace in this section—our goal is to demonstrate ASP.NET in action, so we'll skip over detailed explanations as to how things work behind the scenes. We'll revisit these topics in depth in later chapters.

Setting the Scene

We are going to imagine that a friend has decided to host a New Year's Eve party and that she has asked us to create a web site that allows her invitees to electronically RSVP. She has asked for the following key features:

- A page that shows information about the party and an RSVP form
- Validation for the RSVP form, which will display a confirmation page
- A page that lists the responses from invitees

In the following sections, we'll build on the `PartyInvites` ASP.NET project we created at the beginning of the chapter and add these features.

Creating a Data Model and Repository

Almost all web applications rely on some kind of data model, irrespective of the technology used to create them. We are building a simple application and so we only need a simple data model. Right-click the `PartyInvites` item in the Solution Explorer and select `Add` > `Class` from the pop-up menu.

■ **Tip** If the `Class` menu item is missing or disabled, then you probably left the Visual Studio debugger running. Visual Studio restricts the changes you can make to a project while it is running the application. Select `Stop Debugging` from the `Debug` menu and try again.

Visual Studio displays the `Add New Item` dialog box, which contains templates for all the items you can add to an ASP.NET project. The `Class` template is selected, so set the name to be `GuestResponse.vb` and click the `Add` button. Visual Studio creates a new Visual Basic class file and opens it for editing. Set the contents of the file so that they match Listing 1-4.

■ **Tip** We have used a Visual Basic language feature called *automatically implemented properties* in the `GuestResponse` class, which you may not be familiar with if you have been working with an older version of Visual Basic. We explain the Visual Basic language features that we use in Chapter 3.

Listing 1-4. The GuestResponse Class

```
Public Class GuestResponse
    Public Property Name() As String
    Public Property Email() As String
    Public Property Phone() As String
    Public Property WillAttend() As Nullable(Of Boolean)
End Class
```

■ **Tip** Notice that we have defined the `WillAttend` property as `Nullable(Of Boolean)`. This means that the property can be `True`, `False` or `Nothing`. We'll explain why we chose this data type in the "Performing Validation" section later in the chapter.

We will use instances of the GuestReponse class to represent responses from our party guests. We need a repository to store the GuestResponse objects we create. In a real application, this would typically be a database. We will show you how to set up and use a database in Chapter 6, when we create a more realistic ASP.NET application. In this chapter, we just want something quick and simple, so we are going to store the objects in memory. This is easy to do, but our data will be lost each time that the application is stopped or restarted. This would be an odd choice to make for a real web application, but it's fine for our purposes in this chapter. To define the repository, add a new class file to the project called ResponseRepository.vb and ensure that the contents of the file match those shown in Listing 1-5.

Listing 1-5. The ResponseRepository Class

```
Imports System.Collections.Generic
Public Class ResponseRepository
    Private Shared repository As New ResponseRepository()
    Private responses As New List(Of GuestResponse)()

    Public Shared Function GetRepository() As ResponseRepository
        Return repository
    End Function

    Public Function GetAllResponses() As IEnumerable(Of GuestResponse)
        Return responses
    End Function

    Public Sub AddResponse(response As GuestResponse)
        responses.Add(response)
    End Sub

End Class
```

A repository usually has methods for creating, reading, updating, and deleting data objects (known collectively as CRUD methods), but we only need to be able to read all the data objects and add new ones in this application. We'll show you a more typical repository in Chapter 6.

Creating and Styling the Form

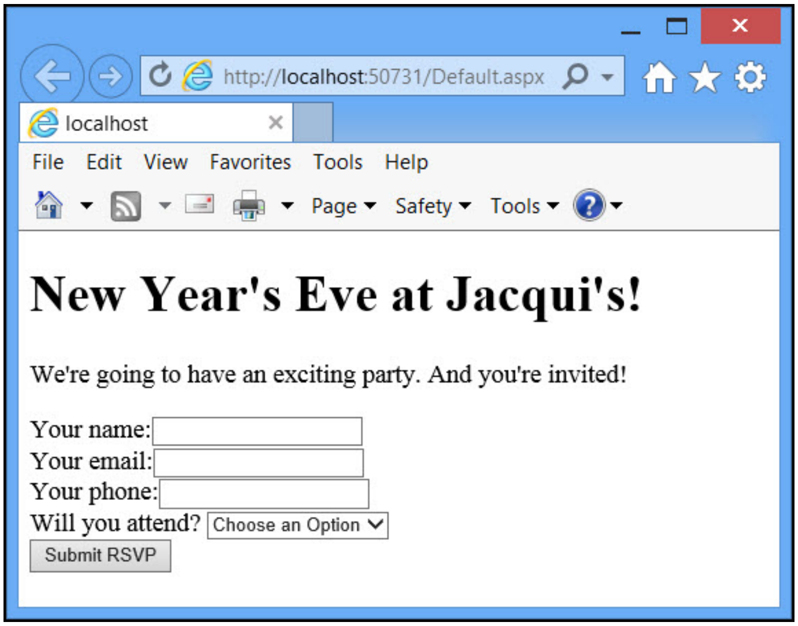
Our next step is to create the page that contains information about the party and an HTML form that allows guests to respond. We will use the Default.aspx file that we created earlier in the chapter. You can see the changes we have made in Listing 1-6.

Listing 1-6. Creating the Form

```
<%@ Page Language="vb" AutoEventWireup="false"
CodeBehind="Default.aspx.vb" Inherits="PartyInvites._Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
```

```
<body>
    <form id="rsvpform" runat="server">
        <div>
            <h1>New Year's Eve at Jacqui's!</h1>
            <p>We're going to have an exciting party. And you're invited!</p>
        </div>
        <div><label>Your name:</label><input type="text" id="name" /></div>
        <div><label>Your email:</label><input type="text" id="email" /></div>
        <div><label>Your phone:</label><input type="text" id="phone" /></div>
        <div>
            <label>Will you attend?</label>
            <select id="willattend">
                <option value="">Choose an Option</option>
                <option value="true">Yes</option>
                <option value="false">No</option>
            </select>
        </div>
        <div>
            <button type="submit">Submit RSVP</button>
        </div>
    </form>
</body>
</html>
```

We have changed the `id` attribute value of the `form` element and added some standard HTML elements to display information about the party and gather the RSVP details from the users. You can see how changes appear by starting the application (either select `Start Debugging` from the Debug menu or click the `Internet Explorer` button on the toolbar). As you can see in Figure 1-7, we have a form but it doesn't look very nice.


http://localhost:50731/Default.aspx
localhost
File Edit View Favorites Tools Help
Page Safety Tools

New Year's Eve at Jacqui's!

We're going to have an exciting party. And you're invited!

Your name:
Your email:
Your phone:
Will you attend? Choose an Option
Submit RSVP

Figure 1-7. *The effect of adding to the form element in the Default.aspx file*

We style elements in a Web Form in the same way we would a regular HTML page—by using Cascading Style Sheets (CSS). To add some basic styles to the application, right-click the `PartyInvites` item in the Solution Explorer and select `Add` > `Style Sheet` from the pop-up menu. Set the name to be `PartyStyles` and click the `OK` button. Visual Studio adds a new `PartyStyles.css` file to the project. Set the contents of this new file to match the CSS shown in Listing 1-7. Although these are very basic CSS styles, they will improve the appearance of our form fields.

Listing 1-7. The CSS Styles Defined in the PartyStyles.css File

```
#rsvpform label { width: 120px; display: inline-block;}
#rsvpform input { margin: 2px; margin-left: 4px; width: 150px;}
#rsvpform select { margin: 2px 0; width: 154px;}
button[type=submit] { margin-top: 5px;}
```

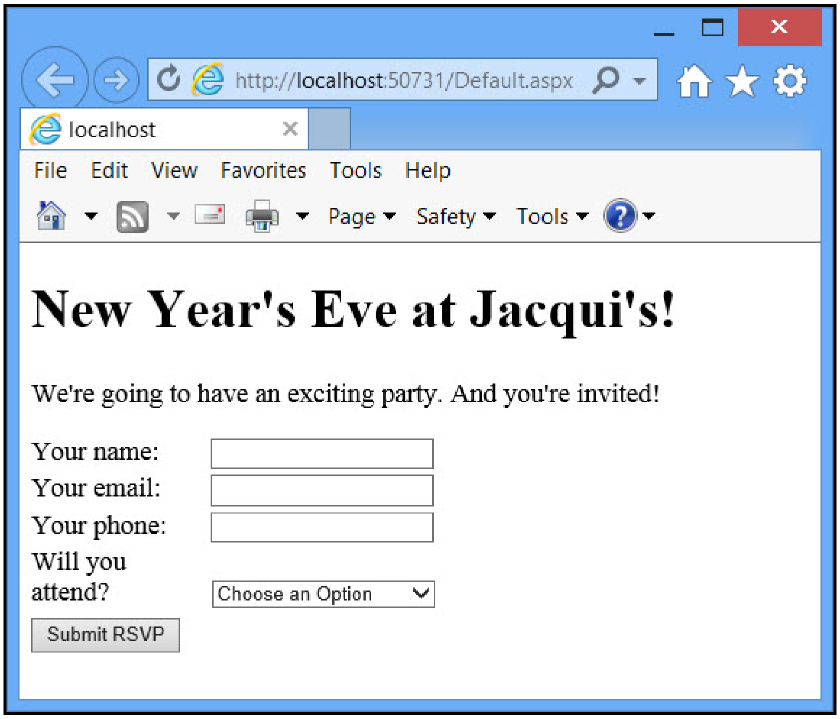
We associate a CSS style sheet with a Web Form using a `link` element. You can see how we have added such an element to the head section of the `Default.aspx` file in Listing 1-8.

■ **Tip** If you are unfamiliar with the standards and technologies that underpin web content, such as HTML, CSS, and basic JavaScript, then we suggest you consult Adam's book *The Definitive Guide to HTML5*, which is also published by Apress and which is a comprehensive reference.

Listing 1-8. Adding a Link Element to the Head Section of the Default.aspx File

```
...
<head runat="server">
    <title></title>
    <link rel="stylesheet" href="PartyStyles.css" />
</head>
...
```

Once again, notice that we are using a standard HTML element to link to a file that contains standard CSS styles. We don't want to labor this point, but one of the nice things about working with ASP.NET is that it builds on your existing knowledge of web standards. You can see the effect of the CSS by starting the application, as illustrated in Figure 1-8.


http://localhost:50731/Default.aspx
localhost
File Edit View Favorites Tools Help
Page Safety Tools

New Year's Eve at Jacqui's!

We're going to have an exciting party. And you're invited!

Your name:
Your email:
Your phone:
Will you attend? Choose an Option

Submit RSVP

Figure 1-8. *The effect of adding a link element for a CSS style sheet to Default.aspx*

Handling the Form

We have a HTML form we can show to people who have been invited to the party, but the same page is displayed over and over again when they click the `Submit RSVP` button. To fix this, we need to implement the code that will handle the form data when it is posted to the server.

At the top of the `Default.aspx` file is the following element:

```
...
<%@ Page Language="vb" AutoEventWireup="false"
CodeBehind="Default.aspx.vb" Inherits="PartyInvites._Default" %>
...
```

This is known as the *Page Directive* and the attributes defined here provide ASP.NET with details about the Web Form file. We'll come back to the directive in detail in Chapter 12, but for now we are interested in the `CodeBehind` attribute. This tells ASP.NET which Visual Basic class file contains the code associated with the Web Form and in this case it is the `Default.aspx.vb` file, which is the code-behind file for `Default.aspx`.

Visual Studio groups related files as a single item in Solution Explorer so that large projects are easier to navigate. If you click the `Show All Files` icon on the toolbar and then click the arrow to the left of the `Default.aspx` entry, you can see the files that Visual Studio has been hiding away. As Figure 1-9 shows, one of them is the `Default.aspx.vb` file referred to by the `CodeBehind` attribute.

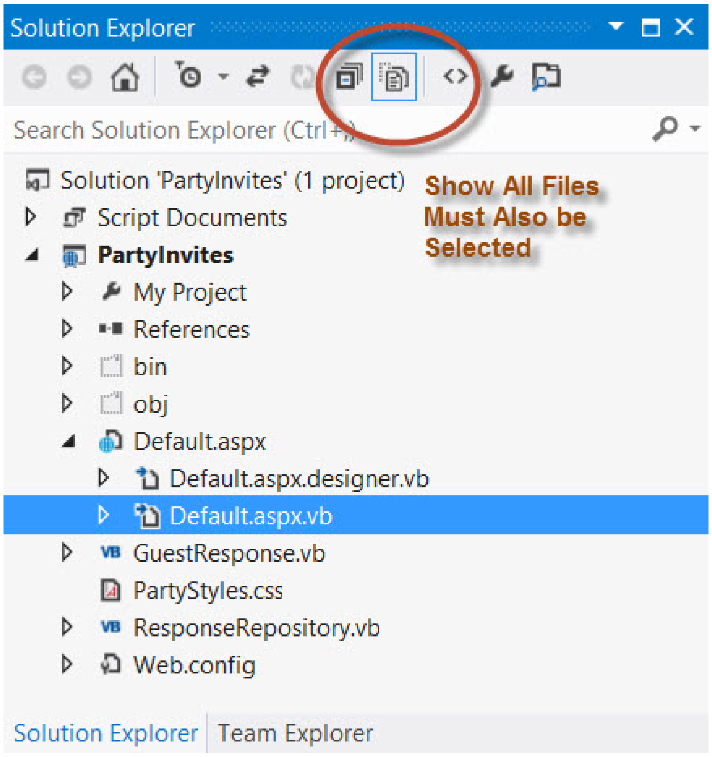


Figure 1-9. *Expanding the Default.aspx file in the Visual Studio Solution Explorer*

Double click the `Default.aspx.vb` file to open it in the editor and you will see the code shown in Listing 1-9. (Extra line breaks are added to keep lines short.)

Listing 1-9. The Initial Content of the Default.aspx.vb Code-Behind File

```
Public Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(
        ByVal sender As Object, ByVal e As System.EventArgs
        ) Handles Me.Load
    End Sub
End Class
```

The base for our code-behind class is `System.Web.UI.Page`, which contains a number of useful methods and properties for responding to web requests. We'll describe the Page class in detail in Part 2 of this book. In this chapter, we are interested in the Page_Load method in our code-behind class that the ASP.NET Framework calls when there are requests for `Default.aspx`, which provides us with the opportunity to response to these requests.

For our example, the Page_Load method will be called once when the page is first loaded and once again when the user submits the form. (We will explain why this happens in Part 2.) In Listing 1-10, you can see the code we have added to the Page_Load method to respond to requests.

Listing 1-10. Adding Code to the Page_Load Method

```
Imports System.Web.ModelBinding
Public Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(
        ByVal sender As Object, ByVal e As System.EventArgs
        ) Handles Me.Load
        If IsPostBack Then
            Dim rsvp As New GuestResponse()
            If TryUpdateModel(
                rsvp,
                New FormValueProvider(ModelBindingExecutionContext)
                ) Then
                ResponseRepository.GetRepository().AddResponse(rsvp)
                If rsvp.WillAttend.HasValue And rsvp.WillAttend.Value Then
                    Response.Redirect("seeyouthere.html")
                Else
                    Response.Redirect("sorryyoucantcome.html")
                End If
            End If
        End If
    End Sub
End Class
```

We determine whether the request we are responding to is the form being posted back to the server by checking the `IsPostBack` property. If it is, we create a new instance of the `GuestResponse` data model object and pass it to the `TryUpdateModel` method, which is inherited from the base `Page` class.

The `TryUpdateModel` method performs a process called *model binding*, where data values from the browser request are used to populate the properties of our data model object. The other argument to the `TryUpdateModel` method is the object that ASP.NET should use to obtain the values it needs—we have used the `System.Web.ModelBinding.FormValueProvider` class, which provides values from form data. We describe model binding in more depth in Part 3, but the result of calling the `TryUpdateModel` method is that the properties of our `GuestResponse` object are updated to reflect the data values that the user submitted in the form. We then store the `GuestResponse` object in our repository.

We want to give the user some kind of feedback when the form is submitted and we do this by using the `Response.Redirect` method, which redirects the user's browser. If the `WillAttend` property is `True`, then the user is coming to the party and we redirect him or her to the `seeyouthere.html` file—otherwise, we redirect the user to the `sorryyoucantcome.html` file.

Creating the HTML Response Files

Not all the pages in an ASP.NET application have to be generated from Web Form files—we can also include regular, static HTML files. To create the first response file, right-click the `PartyInvites` item in the Solution Explorer and select `Add` > `New Item` from the popup menu. Select the `HTML Page` template from the `Add New Item` dialog and set the name to `seeyouthere.html.` Finally, click the `Add` button to create the HTML file. Ensure that the contents of the file match Listing 1-11.

Listing 1-11. The Contents of the seeyouthere.html File

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>See you there!</title>
</head>
<body>
    <h1>See you there!</h1>
    <p>Come around 9pm. Fancy dress is optional</p>
</body>
</html>
```

Repeat the process to create the `sorryyoucantcome.html` file and set the contents to match Listing 1-12.

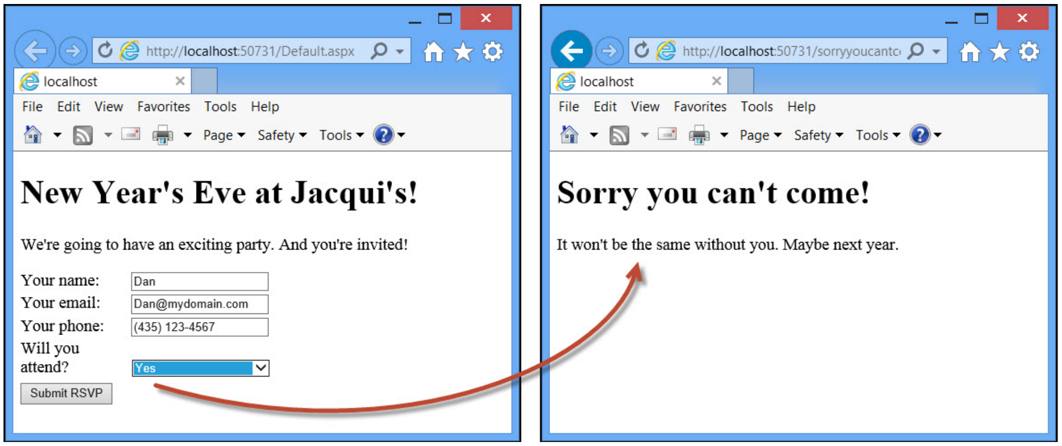
Listing 1-12. The contents of the sorryyoucantcome.html file

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <h1>Sorry you can't come!</h1>
    <p>It won't be the same without you. Maybe next year.</p>
</body>
</html>
```

Bringing the HTML Elements into Scope

We almost have the basic structure of our application in place, but things are not quite working. We need to tell Visual Studio which file should be loaded when we start the application. It didn't matter earlier, because there was only the `Default.aspx` file and Visual Studio is smart enough to figure out that this is the file that we want. But now we have a couple of HTML files as well and we need to give Visual Studio a helping hand. Right-click the `Default.aspx` entry in the Solution Explorer and select `Set as Start Page` from the pop-up menu.

Now you can start the application, either by selecting `Start Debugging` from the `Debug` menu or by clicking the `Internet Explorer` toolbar button. Fill out the form and ensure that you select the `Yes` option from the `select` element. When you submit the form you will see the response that should only be shown when you select the `No` option, as illustrated in Figure 1-10. Clearly, something is a miss.


http://localhost:50731/Default.aspx
localhost
File Edit View Favorites Tools Help
Page ▾ Safety ▾ Tools ▾
Submit RSVP
http://localhost:50731/sorryyoucantc
localhost
File Edit View Favorites Tools Help
Page ▾ Safety ▾ Tools ▾

New Year's Eve at Jacqui's!

We're going to have an exciting party. And you're invited!

Your name: Dan
Your email: Dan@mydomain.com
Your phone: (435) 123-4567
Will you attend? Yes

Sorry you can't come!

It won't be the same without you. Maybe next year.

Figure 1-10. *The application always responds with the negative feedback*

The reason for this problem is that, ASP.NET only looks for elements that have the `runat` attribute with a value of `server` when processing Web Form files. All other elements are ignored and because our `input` and `select` elements in the `Default.aspx` file don't have this attribute/value combination, the model binding process can't find the values submitted in the HTML form. In Listing 1-13, you can see how we have corrected the problem.

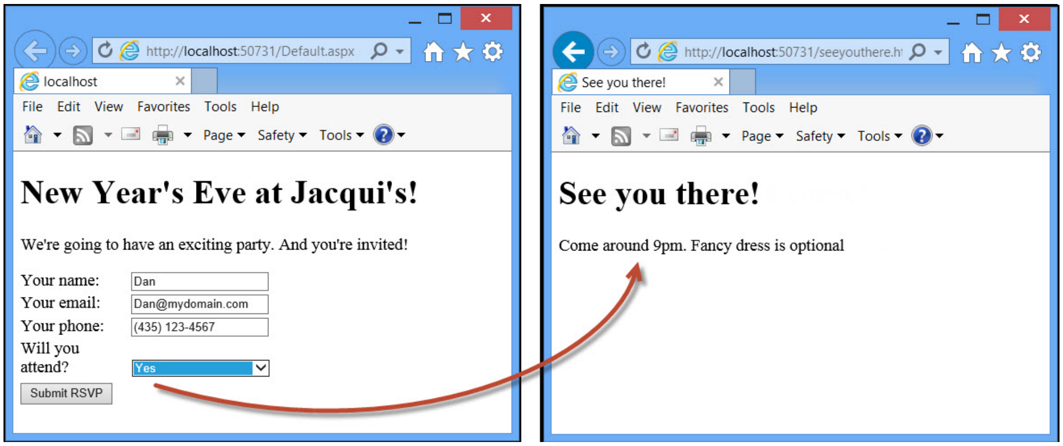
Listing 1-13. Adding the runat Attribute to the Input and Select Elements

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="Default.aspx.vb"
Inherits="PartyInvites._Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link rel="stylesheet" href="PartyStyles.css" />
</head>
<body>
    <form id="rsvpform" runat="server">
        <div>
            <h1>New Year's Eve at Jacqui's!</h1>
            <p>We're going to have an exciting party. And you're invited!</p>
        </div>
        <div><label>Your name:</label><input type="text" id="name" runat="server"/></div>
        <div>
            <label>Your email:</label><input type="text" id="email" runat="server" />
        </div>
        <div>
            <label>Your phone:</label><input type="text" id="phone" runat="server" />
        </div>
        <div>
            <label>Will you attend?</label>
            <select id="willattend" runat="server">
```

```
                <option value="">Choose an Option</option>
                <option value="true">Yes</option>
                <option value="false">No</option>
            </select>
        </div>
        <div>
            <button type="submit">Submit RSVP</button>
        </div>
    </form>
</body>
</html>
```

■ **Tip** There is no value for the `runat` attribute except `server`. If you omit the `runat` attribute or use a value other than `server`, your HTML elements become effectively invisible to ASP.NET. A missing `runat` attribute is the first thing you should check for if your Web Forms are not behaving the way you expect.

Start the application and fill out the form again. This time you will see the correct response when you submit the form, as shown in Figure 1-11.


http://localhost:50731/Default.aspx

New Year's Eve at Jacqui's!

We're going to have an exciting party. And you're invited!

Your name: Dan
Your email: Dan@mydomain.com
Your phone: (435) 123-4567
Will you attend? Yes

Submit RSVP

http://localhost:50731/seeyouthere.h

See you there!

Come around 9pm. Fancy dress is optional

Figure 1-11. *The effect of adding the runat attribute to the input and select elements*

Creating the Summary View

We have the basic building blocks of our application in place and our invitees can RSVP. In this section, we'll add support for displaying a summary of the responses we have received so that our friend can see who is coming and make plans accordingly.

Right-click the `PartyInvites` item in the Solution Explorer and select `Add` > `Web Form` from the pop-up menu. Set the name to be `Summary` and click the `OK` button to create a new file called `Summary.aspx`. Ensure that the contents of this new file match those shown in Listing 1-14.

Listing 1-14. The Contents of the Summary.aspx file

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="Summary.aspx.vb"
    Inherits="PartyInvites.Summary" %>
<%@ Import Namespace="PartyInvites" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <link rel="stylesheet" href="PartyStyles.css" />
</head>
<body>
    <h2>RSVP Summary</h2>
    <h3>People Who Will Attend</h3>
    <table>
        <thead>
            <tr><th>Name</th><th>Email</th><th>Phone</th></tr>
        </thead>
        <tbody>
            <% Dim yesData =
                   ResponseRepository.GetRepository().GetAllResponses().
                   Where(Function(r) r.WillAttend.HasValue AndAlso
                   r.WillAttend.Value)
                For Each rsvp In yesData
                    Dim htmlString As String =
                        [String].Format(
                        "<tr><td>{0}</td><td>{1}</td><td>{2}</td>",
                        rsvp.Name, rsvp.Email, rsvp.Phone)
                    Response.Write(htmlString)
                Next%>
        </tbody>
    </table>
</body>
</html>
```

This is your first ASP.NET application and we want to demonstrate as many techniques as we can into this chapter. This is why the contents of the `Summary.aspx` file look very different from the `Default.aspx` file.

We'll go through the different sections of the file in a moment, but the first thing to notice is that there is no `form` element in the `Summary.aspx` file. The Web Form name is slightly misleading and although forms are useful in most web applications, a Web Form file is really just an enhanced HTML file that is processed by ASP.NET. For the `Default.aspx` file, the enhancements come in the form of the code-behind file so we can use it to deal with form posts. For the `Summary.aspx` file we have gone further and used the `<%` and `%>` tags to add dynamic content to the HTML generated when the browser requests the file.

The official term for the `<%` and `%>` tags is the *service-side scripting delimiters* although they are more commonly referred to as *code nuggets*. The opening tag for a code nugget is just `<%`, without any additional characters. The closing tag for all kinds of code nuggets is always `%>`. There are different kinds of code nuggets available and we added two different types in Listing 1-14. Here is the first one:

```
...
<%@ Import Namespace="PartyInvites" %>
...
```

A code nugget whose opening tag is <%@ is a *directive*. Directives allow you to perform an action that affects the entire Web Form. In this case, we have created an Import directive that brings a namespace from the project into scope so that we can refer to classes without having to qualify the class name. Why do we care about namespaces? Since the other code nugget in the listing is a Visual Basic code block that will be executed when the page is requested, being able to refer to classes without their namespaces makes the code simpler. In particular, Visual Basic won't find this object in another class:

```
ResponseRepository.GetRepository().GetAllResponses()
```

In our code block, we have used regular Visual Basic statements to generate a set of HTML elements that are rows in the table element listing the people who have accepted invitations. We call the ResponseRepository.GetRepository().GetAllResponses() method to get all the data objects in the repository and use the LINQ Where method to select the positive responses. We then use a For Each loop to generate HTML strings for each data object.

```
...
For Each rsvp In yesData
        Dim htmlString As String =
                String.Format("<tr><td>{0}</td><td>{1}</td><td>{2}</td>",
                rsvp.Name, rsvp.Email, rsvp.Phone)
        Response.Write(htmlString)
Next
...
```

The String.Format allows us to compose HTML strings that contain the property values from each GuestResponse object we want to display and we use the Response.Write method to add the HTML to the output sent to the browser.

Formatting the Dynamic HTML

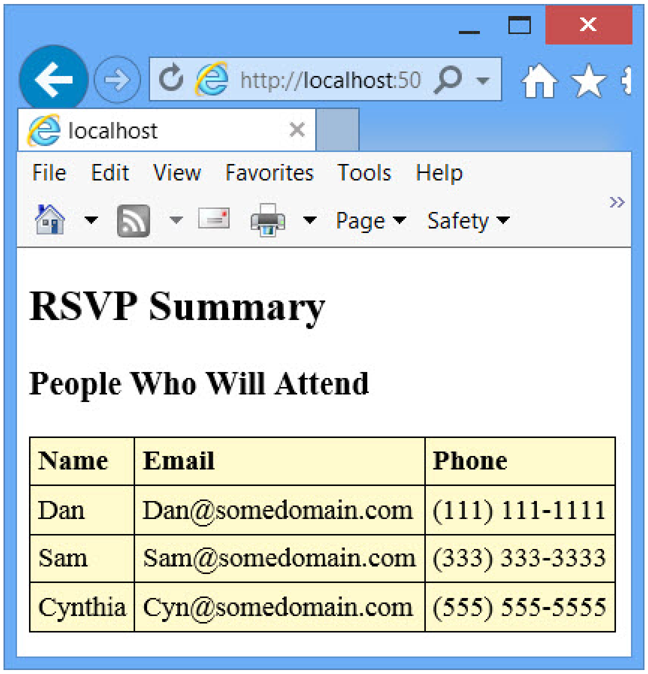
You will notice that we included a link element in the Summary.aspx file that imports the PartyStyles.css file and the styles it contains. We have done this to demonstrate that we style the element that we generate from code blocks in just the same way as the static HTML in the page. In Listing 1-15, you can see the style we added to the PartyStyles.css file for use in Summary.aspx.

Listing 1-15. Adding Styles to the PartyStyles.css File

```
#rsvpform label { width: 120px; display: inline-block;}
#rsvpform input { margin: 2px; margin-left: 4px; width: 150px;}
#rsvpform select { margin: 2px 0; width: 154px;}
button[type=submit] { margin-top: 5px;}

table, td, th {
    border: thin solid black; border-collapse: collapse; padding: 5px;
    background-color: lemonchiffon; text-align: left; margin: 10px 0;
}
```

Testing the Dynamic Code

To test the Summary.aspx file, start the application and use the Default.aspx page to add data to the repository—remember that we are not storing our data persistently in this example and so you need to reenter the data each time you start the application. Navigate to the /Summary.aspx URL once you have submitted the form a few times and you will see the output illustrated in Figure 1-12.



RSVP Summary

People Who Will Attend

Name	Email	Phone
Dan	Dan@somedomain.com	(111) 111-1111
Sam	Sam@somedomain.com	(333) 333-3333
Cynthia	Cyn@somedomain.com	(555) 555-5555

Figure 1-12. *Displaying a summary of the positive replies*

Calling a Code-Behind Method

Although you can include blocks of Visual Basic code in a Web Form file, it usually doesn't make sense to do so because it quickly becomes hard to read and difficult to maintain. A much neater and more common approach is to define methods in the code-behind file and then use a code nugget to call that method and insert the result into the HTML sent to the browser. In Listing 1-16, you can see how we have defined a new method called `GetNoShowHtml` in the `Summary.aspx.vb` code-behind file. This method generates the same kind of table rows we produced in the previous section.

Listing 1-16. The GetNoShowHtml Method in the Summary.aspx.vb Code-behind File

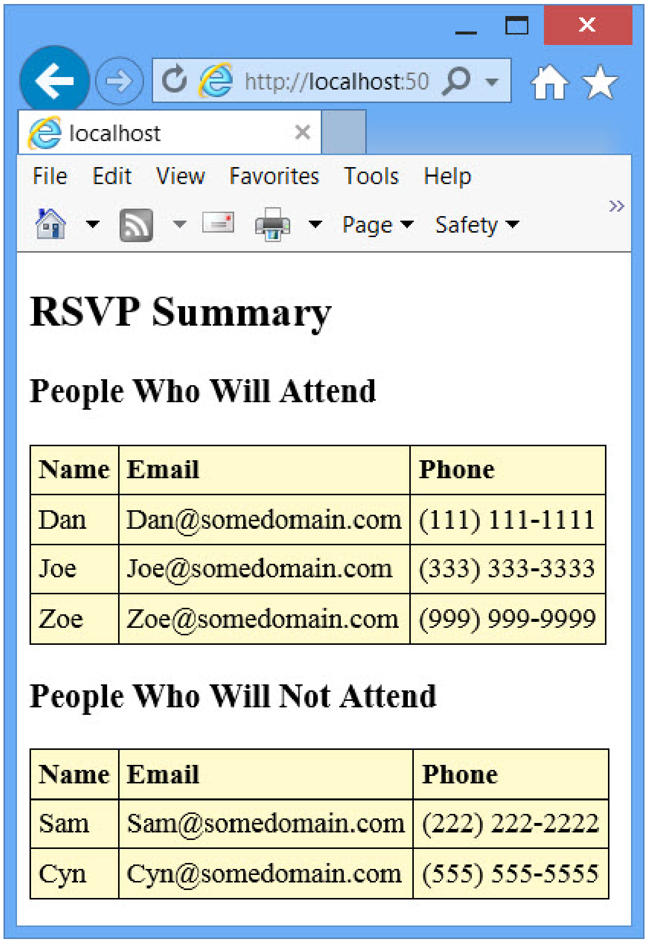
```
Public Class Summary
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(
        ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

    End Sub

    Protected Function GetNoShowHtml() As String
        Dim html As New StringBuilder()
        Dim noData = ResponseRepository.GetRepository().GetAllResponses().
            Where(Function(r) r.WillAttend.HasValue AndAlso Not r.WillAttend.Value)
        For Each rsvp In noData
            html.Append([String].Format(
            "<tr><td>{0}</td><td>{1}</td><td>{2}</td>",
            rsvp.Name, rsvp.Email, rsvp.Phone))
        Next
        Return html.ToString()
    End Function

End Class
```

We can then call this method from a code nugget in the Summary.aspx file, as shown in Listing 1-17.

Listing 1-17. Calling a Code-behind Method from the Summary.aspx File

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="Summary.aspx.vb"
    Inherits="PartyInvites.Summary" %>
<%@ Import Namespace="PartyInvites" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <link rel="stylesheet" href="PartyStyles.css" />
</head>
<body>
    <h2>RSVP Summary</h2>

    <h3>People Who Will Attend</h3>
    <table>
        <thead>
            <tr><th>Name</th><th>Email</th><th>Phone</th></tr>
        </thead>
        <tbody>
            <% Dim yesData =
                   ResponseRepository.GetRepository().GetAllResponses().
                   Where(Function(r) r.WillAttend.HasValue AndAlso
                   r.WillAttend.Value)
                For Each rsvp In yesData
                    Dim htmlString As String =
                        [String].Format(
                        "<tr><td>{0}</td><td>{1}</td><td>{2}</td>",
                        rsvp.Name, rsvp.Email, rsvp.Phone)
                    Response.Write(htmlString)
                Next%>
        </tbody>
    </table>
    <h3>People Who Will Not Attend</h3>
    <table>
        <thead>
            <tr><th>Name</th><th>Email</th><th>Phone</th></tr>
        </thead>
        <tbody>
            <%= GetNoShowHtml() %>
        </tbody>
    </table>

</body>
</html>
```

For this listing we have used the code nugget whose open tag is <%=. This tells ASP.NET to insert the result of the method into the output sent to the browser, which is a neater and more readable approach than including the code directly in the page. The HTML that is generated is the same as for the previous code nugget, except that we are generating table rows for the people who declined their invitation to the party, as shown in Figure 1-13.



RSVP Summary

People Who Will Attend

Name	Email	Phone
Dan	Dan@somedomain.com	(111) 111-1111
Joe	Joe@somedomain.com	(333) 333-3333
Zoe	Zoe@somedomain.com	(999) 999-9999

People Who Will Not Attend

Name	Email	Phone
Sam	Sam@somedomain.com	(222) 222-2222
Cyn	Cyn@somedomain.com	(555) 555-5555

Figure 1-13. *Using a code nugget to insert the result of a method call into the response*

Performing Validation

We are almost finished, but we still have one problem to solve: users can submit any data they want in the Default.aspx form or even post the form without any data at all. We need to make sure that we get values for all the form fields so we have good data and know who is and isn't coming to the party.

ASP.NET provides a range of different validation techniques, but the approach we like best is to apply attributes to the data model class, specifying our validation requirements. We revisit validation in Chapter 8 and cover the topic in depth in Part 3, but you can see how we have applied basic validation to the GuestResponse class in Listing 1-18.

Listing 1-18. Applying Validation Attributes to the GuestResponse Class

```
Imports System.ComponentModel.DataAnnotations

Public Class GuestResponse
    <Required> _
    Public Property Name() As String
    <Required> _
    Public Property Email() As String
    <Required> _
    Public Property Phone() As String
    <Required> _
    Public Property WillAttend() As Nullable(Of Boolean)
End Class
```

The Required attribute, which is in the System.ComponentModel.DataAnnotations namespace, tells ASP.NET that we require a value for the property it is applied to. Because we have applied the attribute to all the properties in the GuestResponse class, we have told ASP.NET that we require properties for all our data model class properties. This is a pretty basic form of validation because we don't check to see whether the value is useful—just that it has been supplied by the user—but it is enough for our example.

■ **Tip** Required is only one of the validation attributes available. We describe the others in Part 3.

When the user submits the form in the Default.aspx file, the ASP.NET Framework will invoke the Page_Load method in the Default.aspx.vb code-behind file. Earlier in the chapter, we showed how we call the TryUpdateModel method to perform model binding. Now that we have added the Required attribute, this method will check to make sure that we have received values for all the properties.

We need to make an addition to the Default.aspx file to display messages to the user when there have been problems validating the form data they have posted. In Listing 1-19, you can see the required addition.

Listing 1-19. Displaying Validation Errors to the User in the Default.aspx File

```
<%@ Page Language="vb" AutoEventWireup="false" CodeBehind="Default.aspx.vb"
Inherits="PartyInvites._Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link rel="stylesheet" href="PartyStyles.css" />
</head>
```