IN THIS CHAPTER

- Getting started with web technology
- Organizing files
- Working with master pages

Online Chapter 3

Digging into Web Construction

You can know the controls, and you can know the framework, but you don't really know the web until you understand the little weird bits, called *glue components*, that make it all work together. No technology has more glue components than web technology.

This chapter is about using some of those glue components to build an application. It gives you a chance to do things that you'll have to do almost every time you make a web application: set up a master page, test your application, and a few other treats.

ASP.NET shields you from a lot of the underlying glue code. You don't have to worry about the details of Common Gateway Interface (CGI), for instance. Nonetheless, there are a number of details that you do have to manage in order to get an application on the Internet, such as the following:

- **Setting up a web project can be . . . demanding.** The default options that Microsoft provides are not immediately obvious, and the need to have them right the first time is high.
- **There are templates for ASP.NET sites.** These templates provide all those things that make a site flow as you navigate through it, like the navigation, advertising, headers, and footers.

- **Security for a web application is . . . different.** Because a significant percentage of web applications is available for public access (quite a shift from Windows applications), you have to consider the reality that people will try to use your application as a platform for phishing, cross-site scripting (XSS), and a hundred other contemporary hacks.
- **Testing web applications is another consideration, because it is more than unit testing.** If your web application goes viral, you might have 1,000,000 visitors tomorrow. You don't have to be ready for that today, but you have to know what you need to do to get ready.
- **Finally, deploying a web application is a new pile of joy that you'll get to know and love.** Remember those Unified Modeling Language (UML) deployment diagrams you learned in school? They can be useful after all! You actually use them in the web world.

Managing Files

Setting up an ASP.NET project is not trivial, and there are a lot of wrong ways to do it. Even though this chapter provides guidelines, the right way depends on your circumstances. Understanding the best practices helps you form your own path. The idea is to know how the tool works in order to help you do your work.

Reviewing the project types

In the book, Chapter 2 of Book 6 discusses the two approaches you can take to create an ASP.NET project in the "Two project approaches" section of the chapter. You see one approach demonstrated in the "Creating a standard project" section and a second approach demonstrated in the "Creating a website" section. Both approaches have advantages and disadvantages, but in the end, the chapter demonstrates that you can use either approach to create the application you need.

Both techniques provide you with access to a number of project types. When you create a new website using the File➪New➪Web Site command, you immediately select an ASP.NET Web Application that relies on the .NET Framework. The File➪New➪Project approach lets you choose from one of three major project types, as described in the "Creating a standard project" section of Book 6, Chapter 2, in the book. With this in mind, different kinds of projects appear in different ways in Visual Studio 2017 within the limits imposed by the individual project types. Table OC3-1 shows all the options.

TABLE OC3-1 **Visual Studio 2017 Project Types for C# 7**

Project Type	Supported Environments	Description
ASP.NET Dynamic Data Entries Web Site	New Web Site	Creates a data-specific website that relies on dynamic-data access techniques using either Entity Framework (EF) or LINQ-to-SQL programming. The structure consists of a presentation layer, a data layer, data-source mapping, and the data store.
ASP.NET Empty Web Site	New Web Site, ASP.NET Web Application (.NET Framework), ASP.NET Core Web Application (.NET Core), ASP.NET Core Web Application (.NET Framework)	Creates an empty website that doesn't include any files but does include basic support.
ASP.NET Web Forms Site	New Web Site, ASP.NET Web Application (.NET Framework)	Creates a basic website that includes standard forms that are ready to fill out and use.
ASP.NET Web Site (Razor v3)	New Web Site	Creates a basic website that includes standard forms. The form language relies on the Razor language (CSHTML) which is based on C#. Essentially, this kind of project is designed to make you more efficient, but only if you're willing to learn the Razor language.
Azure API App	ASP.NET Web Application (.NET Framework)	Constructs a cloud-based application environment that allows public access of services using a Web API. You can publish your APIs so that others can access the code to create a variety of application types.
MVC (Model-View-Controller)	ASP.NET Web Application (.NET Framework)	Creates a basic website based on the Model-View-Controller approach, in which the model encapsulates data, the view displays the data to the user, and the controller encapsulates business logic used to access the data. The goal is to create applications quickly in a manner that is easy to update and works well with teams.
Single Page Application	ASP.NET Web Application (.NET Framework)	Constructs an application environment that is meant to interact with a Web API using a combination of HTML5, CSS3, and JavaScript. The single page represents a starting point that you can use to develop a much larger application.
WCF Service	New Web Site	Defines a method for accessing services by using the Windows Communication Foundation (WCF) framework rather than a more standard Web API. An oversimplification of the differences from the end user (developer access) perspective is that WCF relies on SOAP-based services, while a Web API generally relies on RESTful services.

(continued)

TABLE OC3-1 *(continued)*

Project Type	Supported Environments	Description
Web API	ASP.NET Web Application (.NET Framework), ASP.NET Core Web Application (.NET Core), ASP.NET Core Web Application (.NET Framework)	Defines a method for accessing services by using a Web API rather than WCF. This project type represents the direction that modern applications are taking to simplify code access by using RESTful techniques.
Web Application	ASP.NET Core Web Application (.NET Core), ASP.NET Core Web Application (.NET Framework)	Creates a basic website that relies on the ASP.NET Core approach rather than the full functionality of the .NET Framework. This application can use the MVC methodology of development and includes support for Web API projects (by using the RESTful programming technique).

Organizing files

You have two choices when organizing your website. You can build a few projects to manage your files, or you can build folders into the web project and put your files there. Best practice is to build separate projects for the class files and folders for the web-related files. This way, if you want to reuse the classes for a Windows Mobile application, you can easily just reference the class file project.

Web files should be in folders in the web application. Note that this can have some implications for referencing files. If your project isn't the root of a web server, you might have to carefully consider how to reference files.


TECHNICAL STUFF

If you are deploying your website to a domain, such as `www.mynewWebsite.com`, you can use a `./` to refer to files, like this:

```
<img src="./images/myImage.png">
```

If you aren't deploying to a domain — for instance, if you're making an application that lives in a folder of a site, such as `www.someoneelsessite.com/myApplication` — you can't use `./` because you don't know how far away the root of the application is, or what the directory really is.

ASP.NET can handle this problem for you. A construct in ASP.NET is implemented with the tilde (~), but it works only in ASP.NET controls. That means that the `img` HTML tag won't work with a tilde; you have to use the `asp:Image` control, like this:

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/images/myImage.png"/>
```

The way in which you organize your files can depend on a great many things. Your organization could already have a standard approach that you need to follow. If you're working for a client, you need to use the client's method for organizing the files. However, when starting a new project, it's helpful to use the same organization technique each time. Here is one technique you can use that will make working with your files a lot easier (and less error prone):

- **Put your class libraries in a separate project.** For example, you may have a business logic layer (or controller) in a second project and a data access layer (or model) in a third project when using the MVC approach to application development.
- **Add a** `Bits` folder **that you use to hold all the bits.** For example, your bits folder may include these items:
  - Style sheets
  - JavaScript files
  - Whatever else
- **Create an** `Images` **folder for holding all of the application-specific images.** You may not be a graphic artist, but your application will almost certainly include graphics of some type. By placing your graphics in a separate folder, you make it possible to secure the code, yet grant required access to your graphic artists. In addition, creating the folder enables you to keep those who are working on graphics separate from those who are working on code.
- **Define a root-level folder for every section of the site.** If you have three top-level menu items in your site, create three folders. If you just have application files and don't have a regular menu structure, everything will be in the root.

Mastering Master Pages

Now that there is some form and function to the project, the next step is to add some form and function to the site itself. *Master pages* are exactly what they sound like: structural elements that contain the content of the site as "slave" components.

Master pages are more than page templates. They are fully functional content containers that are supported by Visual Studio in a number of ways. Master pages reduce the complexity of maintaining common content on web pages by

replicating that content automatically and containing the programming to one file from the developer's perspective.

There is no requirement to use master pages. In fact, many of the projects that you'll create won't use master pages because of dependencies on older security or navigational frameworks. That's okay. If you're building something new, certainly consider using master pages.


REMEMBER

If you want to start using master pages in a project from the outset, the fastest method of doing so is to create a Web Forms project. The Web Forms project includes a master page by default, so all you really need to do is modify the code to meet your needs. The following sections assume that you start with an empty project and add a master page to it. This technique is useful when you need to upgrade an existing project to use master pages.

Making a master page

Start by creating a new site. Just follow these steps:

1. **Create a new empty ASP.NET Web application using the steps found in the "Labels versus plain old text" section of Book 6, Chapter 3 of the book and call it** AddMasterPage**.**

   The Empty template creates an empty project that you can use to create whatever sort of project you want.

2. **Right-click the project entry in Solution Explorer and choose Add➪Web Form from the context menu.**

   You see a Specify Name for Item dialog box.

3. **Type** ContentPage **and click OK.**

   Visual Studio adds a new Web Form.

4. **Change to Split view.**

   There will be a default `div` that appears in Book 6, Chapter 2. However, unlike the Web Form in Book 6, Chapter 2, this one won't contain any added tags or text.

Now that you have a shiny new project to use, it's time to add the master page to it. Remember that the goal of this section is to add a master page to an existing Web Form. The following steps show you how:

1. **Right-click the project entry in Solution Explorer and select Add➪New Item from the context menu.**

   You see the Add New Item dialog box.

2. **Select the** `Web\Web Forms` **folder.**

   You see the items displayed in Figure OC3-1. Note that you can create a Web Form with a master page from the outset, which would be the way to do things in a new, empty project under normal circumstances. However, in this case, you want to add a master page to an existing Web Form.

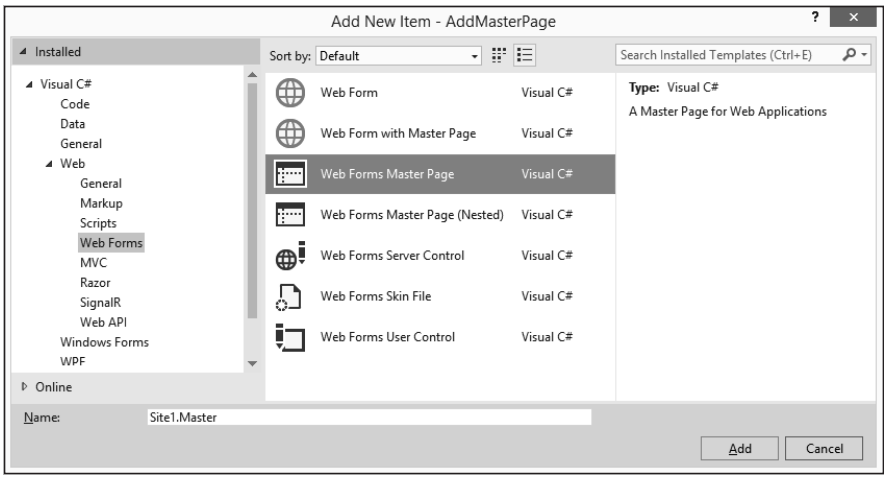

FIGURE OC3-1: Add a new master page to your existing project.

3. **Click Web Forms Master Page.**
4. **Keep the default name of Site1.Master and click Add.**
5. **Change to Split view.**
6. **Open the Properties panel.**
7. **Select the** `<body>` **tag in Source View so that** `DOCUMENT` **is selected in the Properties panel.**
8. **Change the foreground color in the** `Style` **property to read** `color:#333333`**.**
9. **Select the** `<div>` **tag in Source View, press Enter to add a new line between it and the** `<asp:ContentPlaceHolder>` **tag, and type** `<h1>My Website</h1>`**.**

Now you have a template for the rest of your pages. It's a good idea to add navigation here. (This example doesn't add navigation to keep things simple for this introduction.)

Updating existing pages

At this point, you can update an existing page to rely on your master page. The following steps describe what you need to do.

1. **Create a connection between the master page and the Web Form by adding the following attribute to the declaration at the top of the page:**

```
MasterPageFile="~/Site1.Master"
```

   You'll notice that the page now displays a green line under the `<html>` tag and that the Design View says Master Page Error (in rather large letters). That's because the master page acts as a container for the content in the Web Form.

2. **Remove all of the remaining content from the Web Form except any special content in the header and any special content within the `<div>` tags in the body.**

   Given that this is a new project, all you really need to do is remove all the content. However, an existing Web Form will contain content that you want to keep, so performing this process carefully is essential. To use the master page, you need to place your existing content within an `<asp.Content>` tag. There are two: one for the header and a second for the body.

3. **Add the following tags to your Web Form below the declaration:**

```
<asp:Content ID="Content1" ContentPlaceHolderID="head"
    runat="server">

</asp:Content>

<asp:Content ID="Content2"
    ContentPlaceHolderID="ContentPlaceHolder1"
    runat="server">

</asp:Content>
```

WARNING

   The `ContentPlaceHolderID` property values must precisely match the names found in the Site1.Master page. If they don't, you'll receive an error. The property names are part of the connection between the master form and the content form.

4. **Type** This is my body content! **in the area provided by Content2.**

   This paragraph provides content for your page. Normally, you'd move your existing content between the `<asp.Content>` tags as needed.

5. **Choose Debug➪Start Without Debugging.**

   Visual Studio shows you the mix of the master page and the content page, as shown in Figure OC3-2.

FIGURE OC3-2: Master page setups provide both common and specific content as output.

Adding more pages

Obviously, adding master page support to a Web Form after the fact can be complicated. Fortunately, you don't have to go through all that work every time you want to use a master page. The following steps show how to add a master page the easy way:

1. **Right-click the project entry in Solution Explorer and select Add➪New Item from the context menu.**

   You see the Add New Item dialog box.

2. **Select Web Form with Master Page item in the** `Web\Web Forms` **folder.**

3. **Type** ContentPage2.aspx **in the Name field and click Add.**

   The Select a Master Page dialog box, shown in Figure OC3-3, appears.

4. **Click OK.**

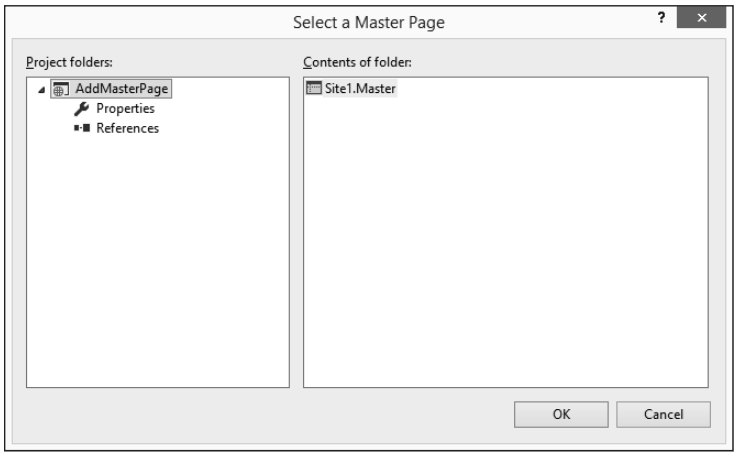

FIGURE OC3-3: Add a new Web Form with master page support already added.

When you add a page using this approach, you automatically get the two `<asp.Content>` tags that you had to add in the previous section. The effect is the same: the two separate areas enable you to add specific content without having to worry about the overall page design.

Deploying Your Masterpiece

Web deployment has historically been difficult in a number of platforms. Because you need the markup files, and any back-end support (like script files or compiled libraries), and perhaps even some middle support, such as an interpreter for your PHP files, getting a web application out to a server can be demanding.

When ASP.NET came out, it promised *xcopy deployment* and, in general, it delivered. Because .NET components are versioned, if you strongly name your DLL files, you can just use the DOS `xcopy` command (or drag and drop in Explorer) to drag them right to your server, thus the term *xcopy deployment*. If you have a permissions thing, you can use File Transfer Protocol (FTP) or whatever.

The first place you should look when you need to deploy an application is the Publish tab of the solution's page, as shown in Figure OC3-4. Note that you can deploy your site to Azure, IIS, FTP, or a folder by using the options listed here. The wizards make the task almost mindlessly simple.

In fact, xcopy deployment has made all sorts of things possible. Dot Net Nuke, now DNN CMS (`http://www.dnnsoftware.com/`), an open source content management system for ASP.NET, has taken things to a logical extreme. When you build a new module for the platform, you add the files to a compressed folder and upload them right through a File Upload control on the site. Then the component is installed and you can use it right away. Neat.

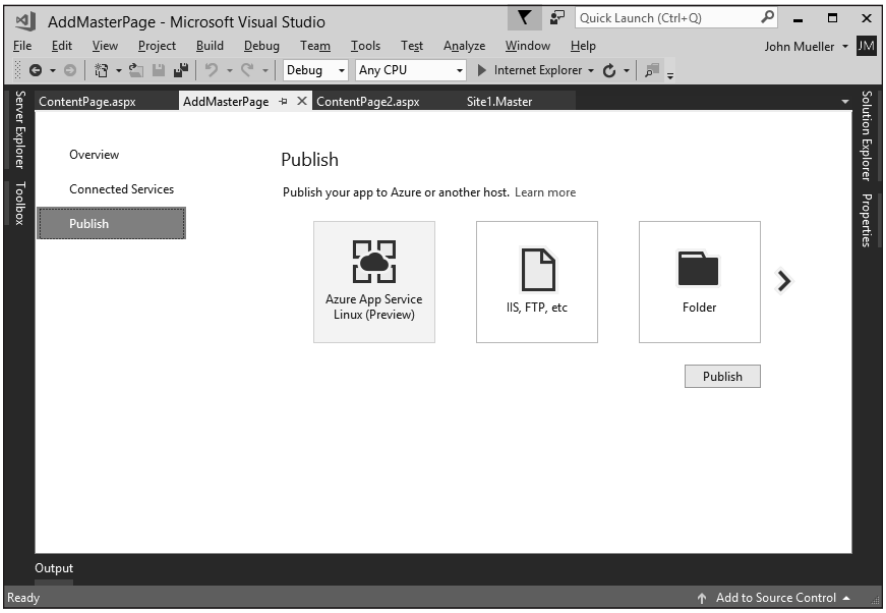

FIGURE OC3-4: The options on the Publish tab provide the easiest deployment options.