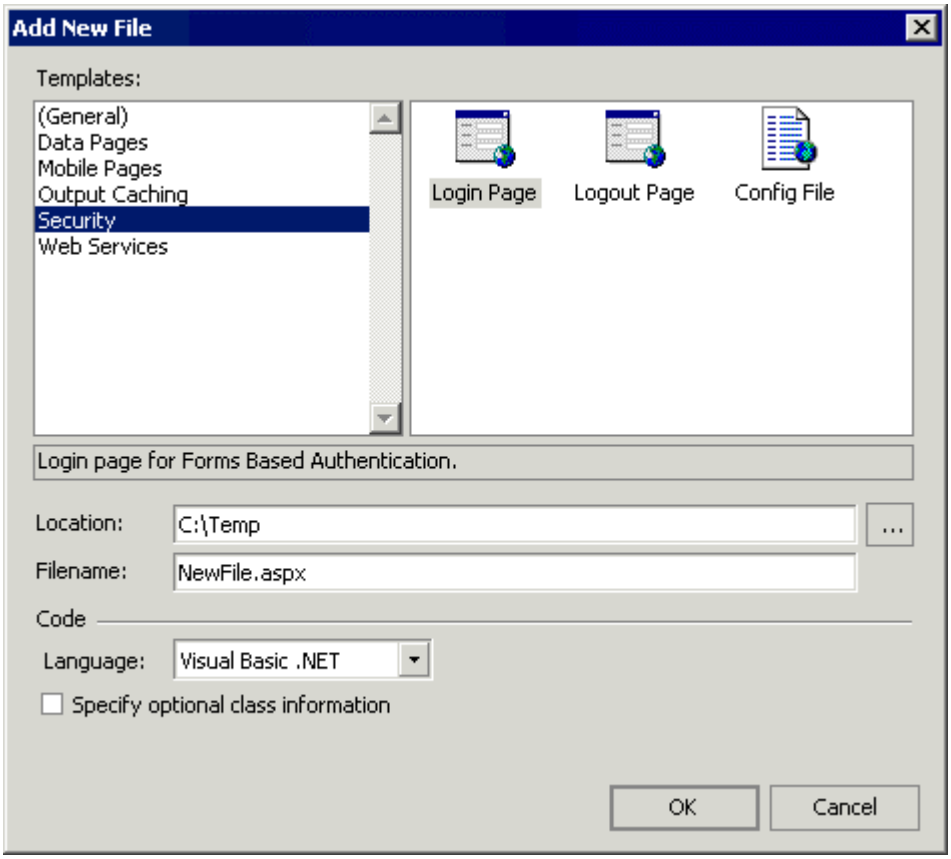
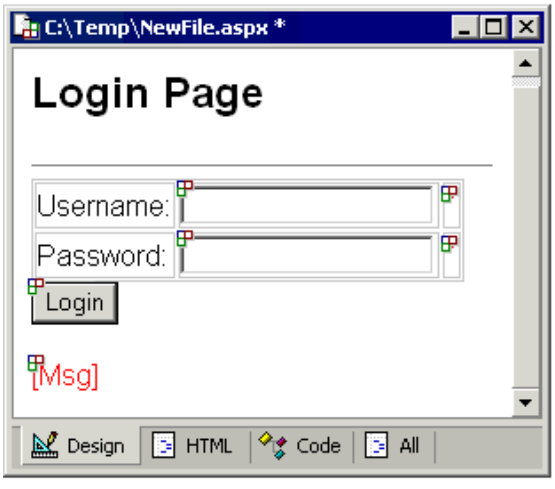
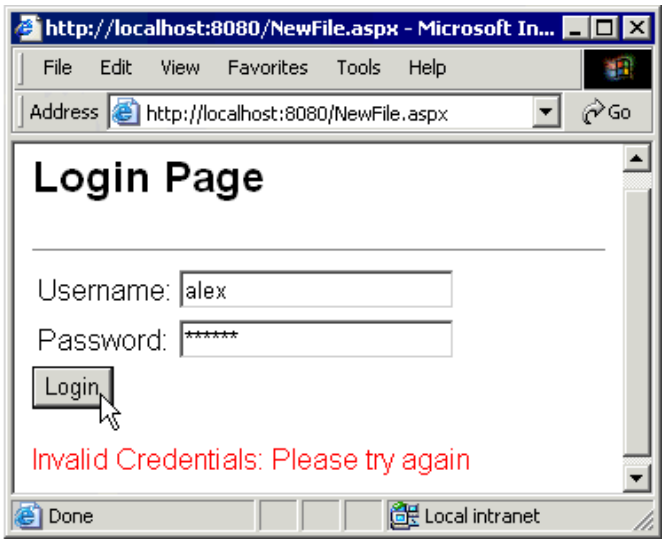
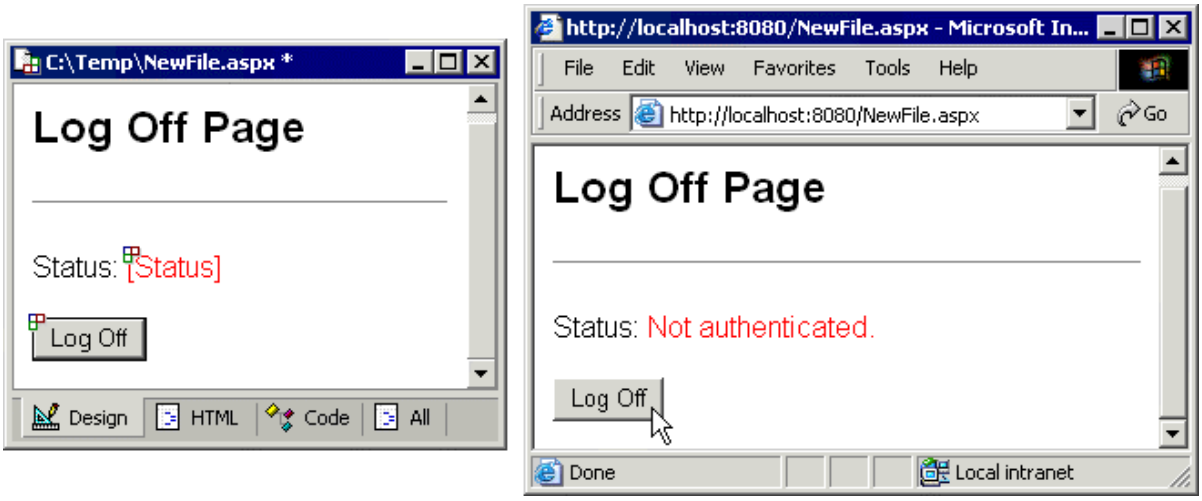


- **Login Page** – this creates a standard "log in" page that contains two textboxes with corresponding `RequiredFieldValidator` controls attached, a `Login` button, and a label where any error message can be displayed. The code in the page uses a simple hard-coded check of the values you enter, and then shows how to execute the `RedirectFromLoginPage` method to load the page that was originally requested. The following screenshots show this page, both in `Design` view and when opened in a browser:





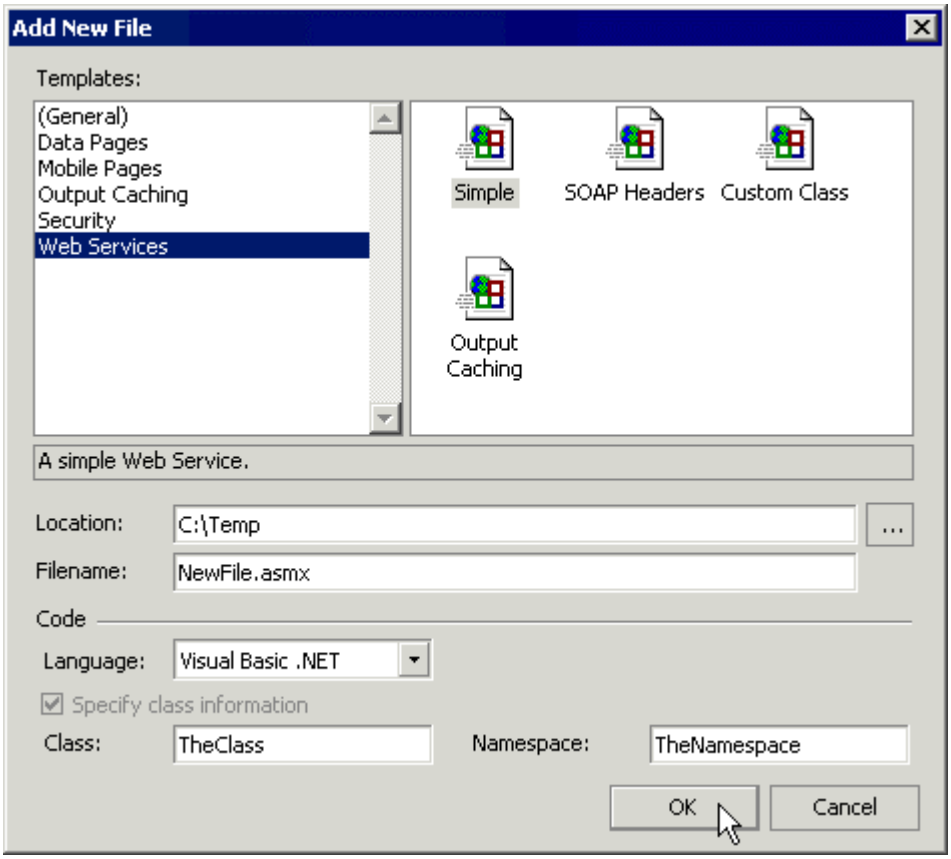
- **Logout Page** – this creates the corresponding "log off" page, with a Status label and a single Log Off button. The label shows the username of the currently logged-in user where available. Clicking the button calls the `SignOut` method and displays a message indicating that the user is no longer authenticated. The following screenshots show the Logout page, in both Design view and when opened in a browser:



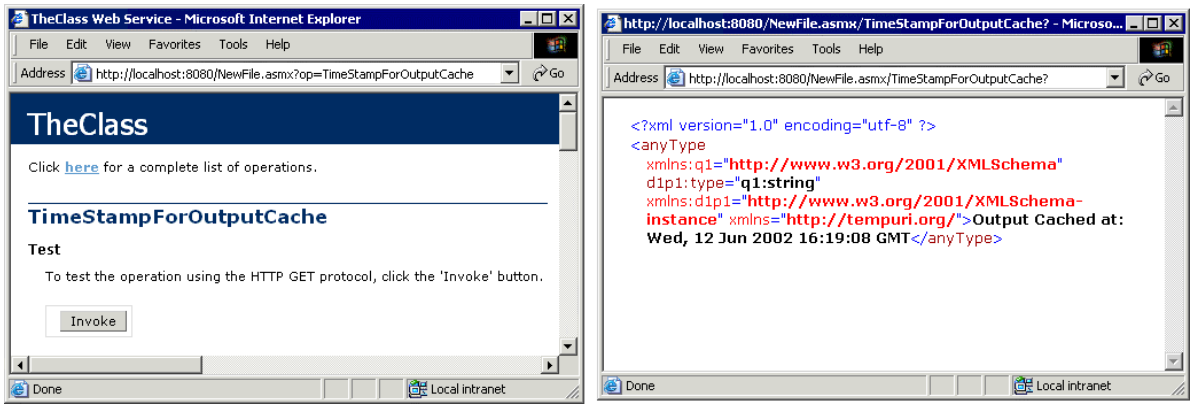
- **Config File** – this example creates a suitable `web.config` file to use with the two previous security examples. The file contains a `<configuration>` element with a child `<system.web>` element. The `<system.web>` element contains the `<authentication>` and `<authorization>` elements that specify Forms authentication, and deny anonymous users.

File Types in the Web Services Section

The final section of the New File dialog is the Web Services section. This includes four example pages that implement different features of Web Services. For each one, you must enter the class name and namespace before you can create the file:



- **Simple** – this example creates the simplest type of Web Service, basically the same as the XML Web Service option in the (General) section of the New File dialog. The file contains an `@WebService` directive, `Imports` or `using` statements for the required Web Service namespaces, a `Class` definition, and an example public function outline that you can modify.
- **SOAP Headers** – this example creates a Web Service that reads a custom value from the SOAP headers of the request, and displays the result.
- **Custom Class** – this example demonstrates how a custom class can be returned from a Web Service. The code creates an instance of a custom class named `OrderDetails` (which is actually an array of another custom class named `OrderItem`), sets some values for the class members, and then returns this instance.
- **Output Caching** – this example demonstrates how the output from a Web Service can be cached, much like the examples shown earlier that used output caching. It simply defines a public function that is implemented as a `WebMethod`, and adds a `CacheDuration` attribute with a value of `30` to the function so that the output is automatically cached for thirty seconds. The following screenshots show the page opened in a browser, and the result:



Language, Class Names, and Namespaces

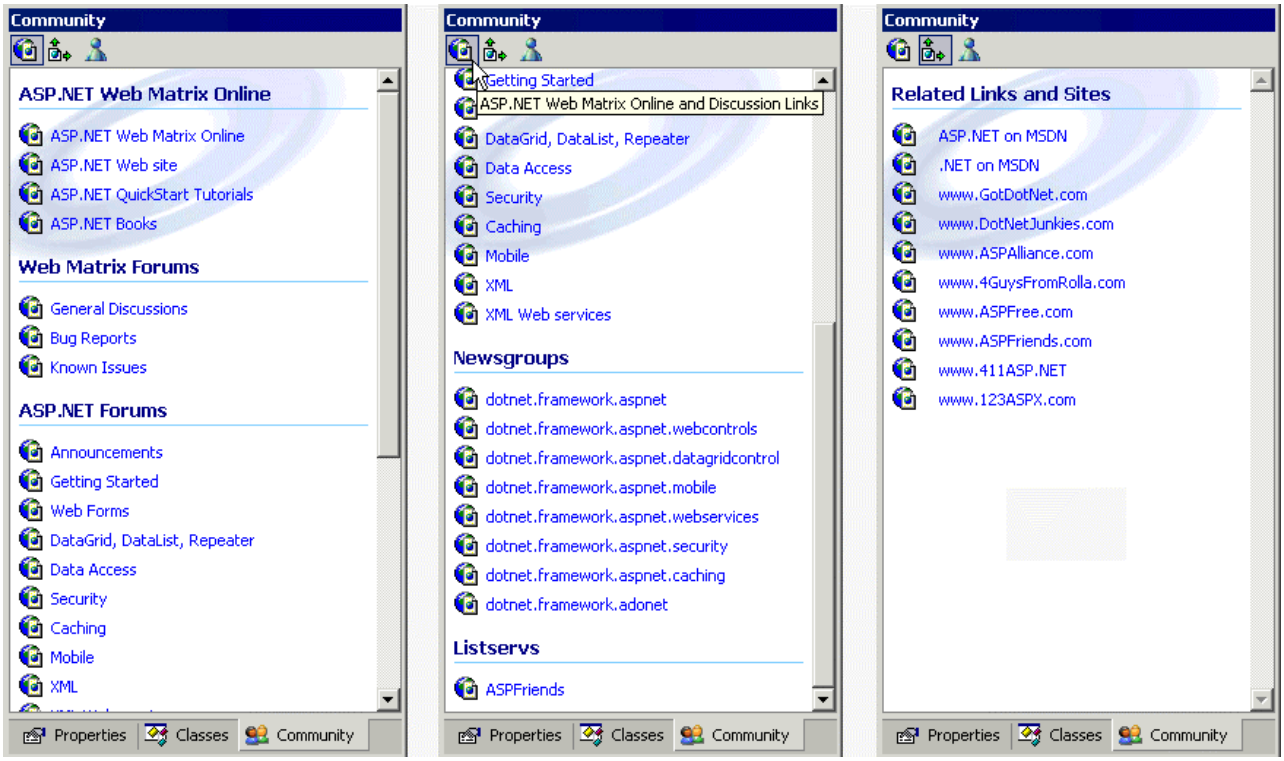
Remember that, for each type of file selected in the New File dialog, any code automatically included in the file is in the language that you specify in that dialog – the choice is between Visual Basic .NET and C#. Depending on the type of file you select, the dialog will also contain controls in which any other required information is entered – such as the class name and namespace (in some cases this is optional, while in others – such as a Web Service or Class file – it is mandatory). We'll create some example pages later on in order to demonstrate these general techniques.

Help, Support, and Reference Information

We've seen how Web Matrix provides access to reference materials and online help in several ways. Future plans are for Web Matrix to include its own comprehensive help files that describe the workings of the IDE, and how to get the best from the product. Only minimal built-in help features are currently implemented at the moment, such as the links to various resources and samples at http://www.asp.net/ and http://www.gotdotnet.com/. However, if you place the cursor over a class name in the Edit window and press the *F1* key, a new ClassBrowser window opens with reference details of that class.

Several other places within the Web Matrix IDE also provide access to online help and support. The Community window (in the lower part of the "project" window) contains links to the ASP.NET Web Matrix site, as well as links to several Microsoft-run .NET newsgroups, and list servers provided by other members of the Web Matrix community.

The ASP.NET Web Matrix site is part of the main ASP.NET site at http://www.asp.net/WebMatrix, which also contains a great deal of useful information and links to other ASP.NET-related sites. It is also the prime source for downloadable add-ins, control libraries and other resources for Web Matrix – including access to the latest version of the product. Two views of the first page follow so that you can see the range of resources that are provided:



The second page (opened from the second icon at the top of the Community window) contains links to related web sites and other resources, while the third page (opened from the third icon) accesses MSN Messenger (if you have this installed on your machine), so that you can chat in real time with other Web Matrix users.

Don't forget that the main toolbar at the top of the Web Matrix window contains a combobox drop-down list in which you can type a question or a series of keywords. Pressing *Return* opens the ASP.NET Web Matrix site in your default browser, and displays a list of articles and resources that match your query.

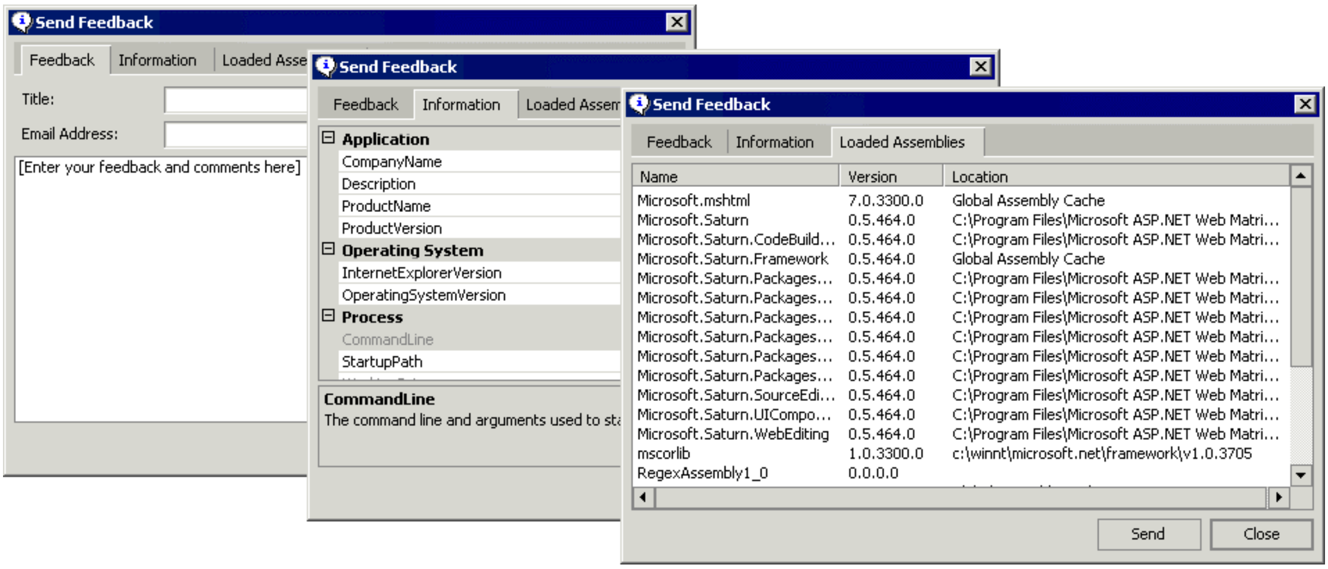
You'll also recall from our earlier discussion that the ClassBrowser window contains links for each member of the .NET Framework Class Library that open the corresponding help and reference pages either locally from your own machine, or at the MSDN online library site.

Sending Feedback to Microsoft

The Help menu in the Web Matrix IDE contains an entry to send feedback on the product to Microsoft. This feedback can consist of bug reports, feature requests, or just general information and comments.

> **Web Matrix is a "community product", and, as such, its future development will be guided to a large extent by the feedback Microsoft receives from users. So, don't be afraid to send in your opinions – the development team is keen to hear what you think!**

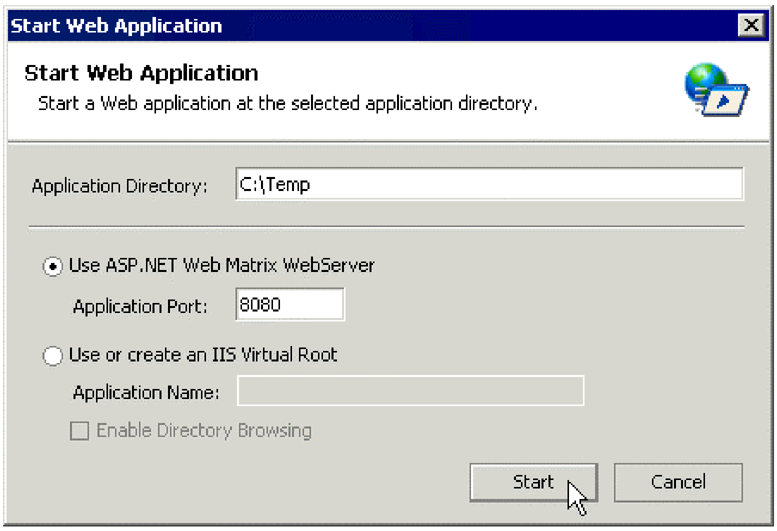
The Send Feedback window is a three-page tabbed dialog that contains the Feedback page itself, the application Information page, and a list of all the currently Loaded Assemblies (the same dialog, but without the Feedback page, appears when you select the Application Information command from the Help menu):



The Microsoft ASP.NET Web Matrix Web Server

Before we move on to *Part 2*, where we'll see Web Matrix in action, we'll take a quick look at the web server that is provided with Web Matrix. This is a slim and lightweight web server that can be used to run ASP.NET pages and other resources (such as web Services) on machines that do not already have a local Web server installed.

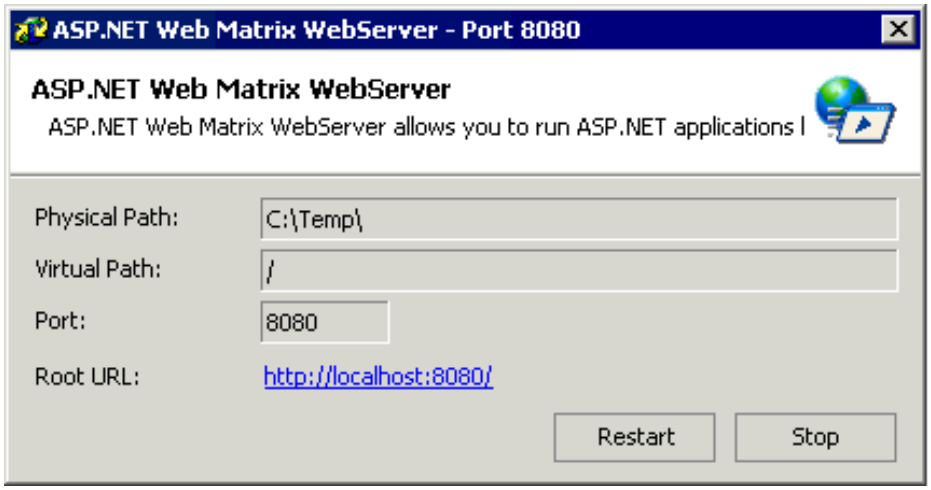
When you first run an ASP.NET page or Web Service from within the Web Matrix IDE, a dialog opens that asks you which web server you want to use. As shown in the following screenshot, you can allow the Microsoft ASP.NET Web Matrix Web Server to execute your page or Web Service:

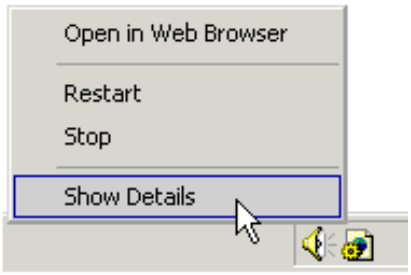


As you can see, the default for the web server is to run on port `8080`. This is ideal if the machine you are using already has a web server (such as IIS) installed and running. The existing Web server is likely to be using port `80`, and so by using a different port the Web Matrix Web server avoids any possibility of an error. You can change the Application Port to a different port if you wish (such as port `80` if you don't have IIS installed).

Alternatively, you can select an existing instance of Internet Information Server (IIS) to execute your page, in which case Web Matrix will create a new virtual root (with the name you specify) that points to the folder containing the file you are editing. If you wish, you can also turn on directory browsing for this virtual root, which makes it easier to find and run individual pages as you develop your application.

Once the Web Matrix Web server has been started, an icon appears in the Windows Taskbar tray. Right-clicking this icon displays a menu in which you can open the web site that the web server is providing in your default browser, Restart or Stop the web server, or show details about it:



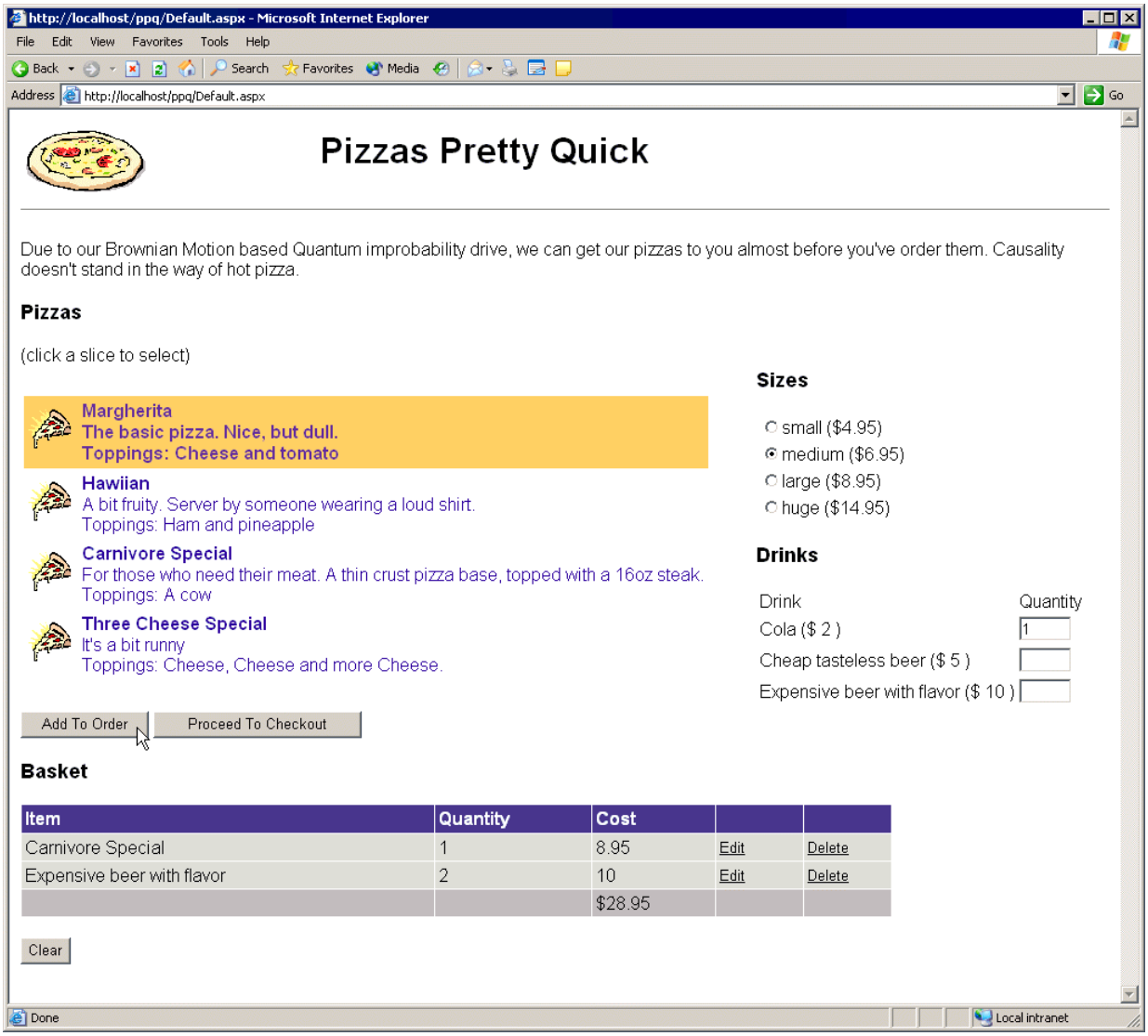


Part 2 – Putting Web Matrix to Work

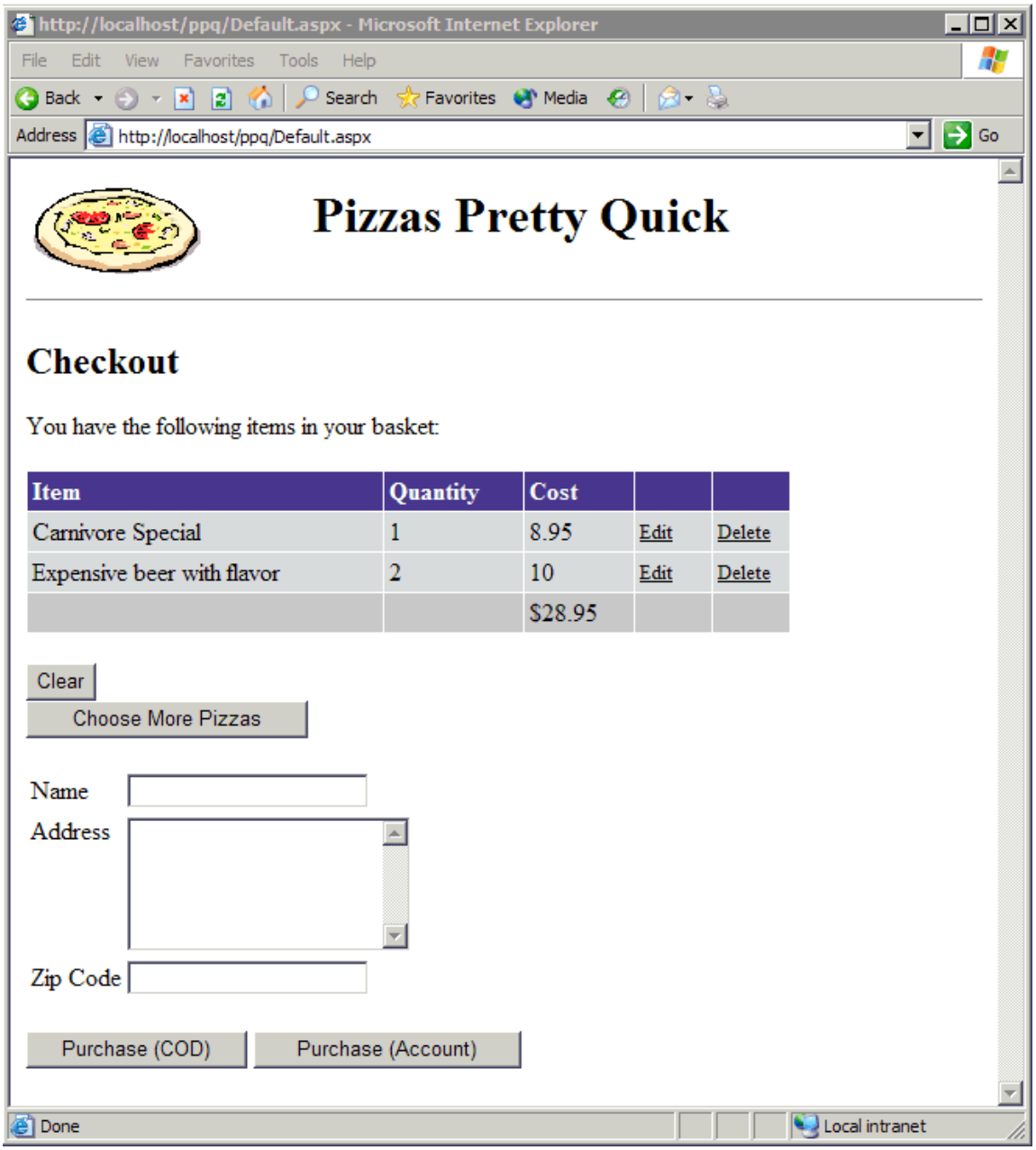
Now that you know what tasks Microsoft ASP.NET Web Matrix is capable of, it's time to put them into practice. Web Matrix is easy to use, so we're not going to show you every aspect of it in action. Instead, we'll build a simple web site for a pizza delivery company, concentrating on the most commonly used pages. This will show you just how little you need to do to get a site up and running with Web Matrix.

Pretty Quick Pizza

Our sample web site is designed to allow customers to pick pizzas and drinks, add them to a simple shopping basket, and then proceed to a checkout where they pay either by cash on delivery or by being billed to an account. It's a really simple e-commerce site, and leaves out many features (such as looking cool!) because they aren't required. We'll end up with a simple site like this:



From this page a customer can select from a variety of types and sizes of pizza and from a range of drinks. Their selection can be added to a shopping basket, and once the customer has made all their choices, they can proceed to the checkout:



The checkout page redisplays the customer's selection and allows the order to be placed. The customer can choose to pay when the pizza is delivered or to have the amount billed to an account. If the customer chooses to have the amount billed to an account they will be taken to a secure login page where they can access their account details.

> **All of the code for this example is available from http://www.AlAndDave.com/books/webmatrix/.**

Building ASP.NET Pages

Because we're not building a fully functional site, we've cut out some of the stuff that you'd normally use. For example, we've only got a minimal data access layer, limited security, and few advanced features. This is because what's important is showing you the types of pages Web Matrix can create, and what you need to do to customize them for your own requirements.

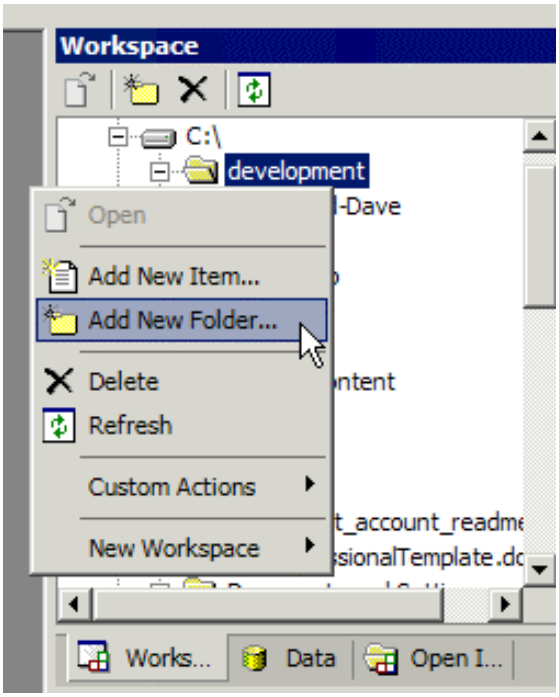
As Web Matrix is file based, you'll need to set up the IIS Virtual Directory yourself. I called it PPQ.

In the following sections we'll tackle:

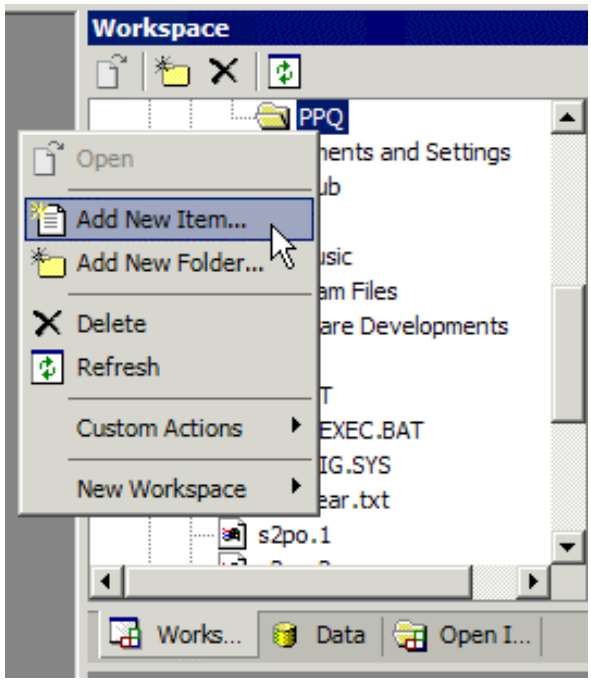
- Creating a Data Layer that consists of an XML file, a VB.NET component, and some SQL stored procedures and tables
- Creating User Controls for the page header and the shopping basket
- Creating the pizza selection page, where we use a variety of ASP.NET controls, as well as the newly created User Controls
- Creating the checkout page, where we take the user's details and how they'd like to pay
- Creating secure pages for customers with accounts
- Creating a variety of different pages, such as those that user a Master and Details grid, or those that require caching
- How to create Web Services
- How to use other controls, such as the Internet Explorer controls and custom controls

The Data Layer

The data layer for this application consists of two files: an XML file that contains the data and a class that loads the data and performs some database logging. When you first start Web Matrix you'll be presented with the New File dialog (remember that Web Matrix is file based, and not project based like Visual Studio .NET). To keep the files for this web site together we'll need to create a directory – we can do this either externally in Explorer, or from within Web Matrix using the Workspace, where we select New Directory from the context menu:



Once we've created the directory, we need to start creating the files for the application. First of all, we need to create the XML file. You can do this from the New … item on the File menu, or use the context menu on the directory:



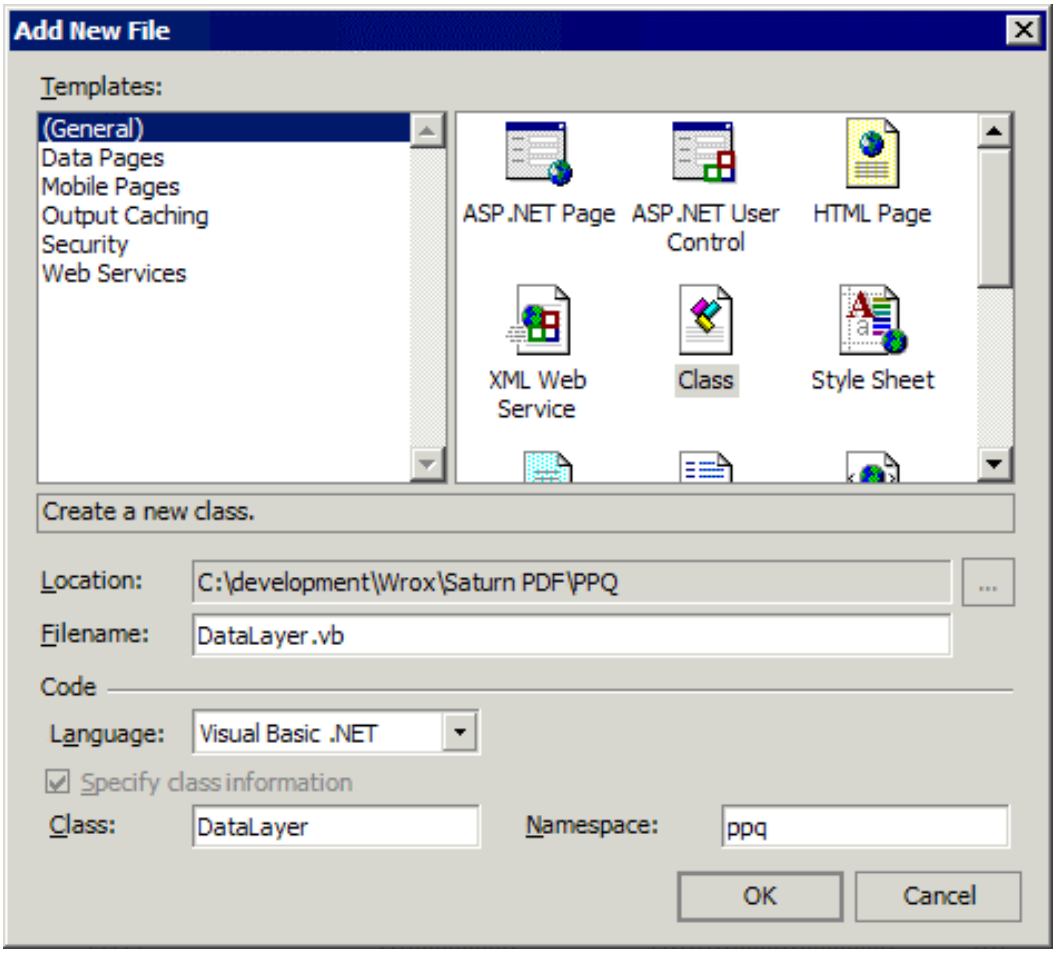
This brings up the New File dialog, from where you can select XML File from the General templates. You then need only add your data:

C:\development\Wrox\Saturn PDF\PPQ\pizzas.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<ppq>
    <pizza name="Margherita" ingredients="Cheese and tomato"
           description="The basic pizza. Nice, but dull."/>
    <pizza name="Hawiian" ingredients="Ham and pineapple"
           description="A bit fruity. Server by someone wearing a loud shirt."/>
    <pizza name="Carnivore Special" ingredients="A cow"
           description="For those who need their meat. A thin crust pizza base, topped with a 16oz steak."/>
    <pizza name="Three Cheese Special" ingredients="Cheese, Cheese and more Cheese."
           description="It's a bit runny"/>
    <drink name="Cola" price="2"/>
    <drink name="Cheap tasteless beer" price="5"/>
    <drink name="Expensive beer with flavor" price="10"/>
    <size name="small ($4.95)" price="4.95"/>
    <size name="medium ($6.95)" price="6.95"/>
    <size name="large ($8.95)" price="8.95"/>
    <size name="huge ($14.95)" price="14.95"/>
</ppq>
```

In reality it's likely that you'd use a database for all of these details, but this is a quick solution that shows the simplicity of Web Matrix – note that there are no special XML editing features, such as XML validation, that would add complexity to the tool.

To use this data, we create a class called `DataLayer`:



Here we have the option to select the default language, the class name, and the namespace for the class. The template created is a stub into which you add your required code. We could have used the Insert Data Method code builder to add code but the code generated by the code builder creates a SQL statement to execute, and we want to use a couple of stored procedures. We'll add the code manually (although you could still use the code builder and then modify the generated code):

```
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Web
Imports System.Xml

Namespace ppq

  Public Class DataLayer

    Public Sub New()
    End Sub

    Public Shared Function GetData() As DataSet

      Dim ctx As HttpContext = HttpContext.Current
      Dim ds As New DataSet()
```

```
      ds.ReadXml(ctx.Server.MapPath("pizzas.xml"))
      Return ds

    End Function

    Public Shared Sub LogOrder(Name As String, Address As String, _
                               ZipCode As String)

      Dim ctx As HttpContext = HttpContext.Current
      Dim Basket As DataTable = CType(ctx.Session("Basket"), DataTable)

      Dim conn As New SqlConnection("server=.; " & _
                        "DataBase=AlandDave; Trusted_Connection=true")
      conn.Open()

      ' add the order
      Dim cmd As New SqlCommand()
      cmd.Connection = conn
      cmd.CommandText = "sp_PPQInsertOrder"
      cmd.CommandType = CommandType.StoredProcedure

      cmd.Parameters.Add("@Name", SqlDbType.VarChar, 25).Value = Name
      cmd.Parameters.Add("@Address", SqlDbType.VarChar, 255).Value = Address
      cmd.Parameters.Add("@ZipCode", SqlDbType.VarChar, 15).Value = ZipCode

      dim OrderID As Integer = cmd.ExecuteScalar()


      ' add the order details
      cmd.Parameters.Clear()
      cmd.CommandText = "sp_PPQInsertOrderItem"
      cmd.Parameters.Add("@fkOrderID", SqlDbType.Int)
      cmd.Parameters.Add("@Item", SqlDbType.VarChar, 25)
      cmd.Parameters.Add("@Quantity", SqlDbType.Int)
      cmd.Parameters.Add("@Cost", SqlDbType.Decimal)

      cmd.Parameters("@fkOrderID").Value = OrderID
      Dim row As DataRow
      For Each row In Basket.Rows
        cmd.Parameters("@Item").Value = row("Description")
        cmd.Parameters("@Quantity").Value = row("Quantity")
        cmd.Parameters("@Cost").Value = row("Cost")
        cmd.ExecuteNonQuery()
      Next

      conn.Close()

    End Sub

  End Class

End Namespace
```