



# Inside ASP.NET Web Matrix

Alex Homer and Dave Sussman



online discussion at: [p2p.wrox.com](http://p2p.wrox.com)  
for more information on Wrox books visit: [www.wrox.com](http://www.wrox.com)

# **Inside ASP.NET Web Matrix**

Alex Homer  
Dave Sussman

*Wrox Press Ltd.* ®

# **Inside ASP.NET Web Matrix**

© 2002 Wrox Press

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

The author and publisher have made every effort during the preparation of this book to ensure accuracy of the material. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, Wrox Press, nor its dealers or distributors will be held liable for any damages caused or alleged to have been caused either directly or indirectly by this book.



Published by Wrox Press Ltd,  
Arden House, 1102 Warwick Road, Acocks Green,  
Birmingham, B27 6BH, UK

## **Trademark Acknowledgements**

Wrox has endeavored to adhere to trademark conventions for all companies and products mentioned in this book, such as the appropriate use of capitalization. Wrox cannot however guarantee the accuracy of this information.

## **Credits**

### **Authors**

Alex Homer  
Dave Sussman

### **Managing Editor**

Viv Emery

### **Commissioning Editor**

Daniel Kent

### **Production Coordinator &**

#### **Cover**

Natalie O'Donnell

### **Technical Editor**

Daniel Richardson

## About the Authors

**Alex Homer** is a computer geek and Web developer with a passion for ASP.NET. Although he has to spend some time doing real work (a bit of consultancy and training, and the occasional conference session), most of his days are absorbed in playing with the latest Microsoft Web technology and then writing about it. Living in the picturesque wilderness of the Derbyshire Dales in England, he is well away from the demands of the real world – with only an Internet connection to maintain some distant representation of normality. But, hey, what else could you want from life?

You can contact Alex through his own software company, Stonebroom Limited: [alex@stonebroom.com](mailto:alex@stonebroom.com).

**Dave Sussman** is a hacker in the traditional sense of the word. That's someone who likes playing with code and working out how things work, which is why he spends much of his life working with beta software. Luckily this coincides with writing about new technologies, giving him an output for his poor English and grammar. He lives in a small village in the Oxfordshire countryside. Like many programmers everywhere he has an expensive hi-fi, a big TV and no life.

You can contact Dave through his own company, Ipona Limited: [davids@ipona.co.uk](mailto:davids@ipona.co.uk).

# Inside ASP.NET Web Matrix

During its relatively short but spectacularly successful life, Microsoft® Active Server Pages (ASP) has grown from a simple scripting environment for creating dynamic Web pages into a powerful and easy-to-use platform for fully-fledged Web application development. In its latest incarnation, ASP.NET, it provides a complete solution for building almost any type of interactive user interface, as well as for implementing extensive back-end processing operations.

However, despite the many powerful features of ASP, choosing a comprehensive and usable development environment in which to create ASP applications was never easy. Many third parties provide ASP support in their products, for example HomeSite and Macromedia UltraDev (amongst others) support ASP 3.0, and, of course, Microsoft's own Visual Studio 6.0 included InterDev – which was also available as a stand-alone product.

With the advent of .NET, support for ASP.NET development has been fully integrated into Visual Studio .NET. It provides an extremely powerful and usable environment for ASP.NET development in the guise of Web Forms, as well as the more traditional types of application (Windows Forms). And now Visual Studio .NET is joined by another Microsoft product, namely the **Microsoft ASP.NET Web Matrix Project** (referred to from here on in as "Web Matrix").

At the time of writing, Web Matrix has just been released as a Beta 1 product. The whole nature of the Microsoft ASP.NET Web Matrix project is that it will develop and grow based on feedback from the community that uses it, so the feature set will evolve over time. You should also keep in mind that, as this is a Beta product, there are quite a few features that are not yet fully implemented (so some things you may expect to see are missing).

However, even at this stage Web Matrix is an extremely usable and efficient tool, and certainly well worth installing and experimenting with. In time, it will, without doubt, mature and be extended to provide many more of the features required for building Web sites and Web applications using ASP.NET.

Over three sections this document will explore what Web Matrix is, what it can do, and how you can use it:

- ❑ *Part 1 – What is Web Matrix?* provides an overview of Web Matrix, looks at the features it provides, and the IDE it contains
- ❑ *Part 2 – Putting Web Matrix to Work* walks you through using Web Matrix to build an application that contains many different types of pages and resources
- ❑ *Part 3 – Configuring and Extending Web Matrix* demonstrates how Web Matrix can be configured to suit your individual requirements, and extended by installing your own or third party add-ins

## Part 1 – What is Web Matrix?

From first impressions, you may think that Web Matrix is just a simplified development environment for building ASP.NET applications. In fact, it provides much more than this. As well as ASP.NET pages (including mobile device pages), Web Matrix can be used to create user controls and class files (for compiling into assemblies), Web service files, and even HTTP Handlers. It also provides integrated support for creating and editing HTML pages, style sheets, XML schemas and documents, text files and SQL scripts, and .NET configuration files (such as `web.config` and `global.asax`).

Web Matrix also provides powerful wizards that automate much of the process of creating pages that handle data, pages that use output caching, and pages that use the built-in ASP.NET authentication features. It also comes complete with its own Web server, and other useful add-ins. You can even create and install your own add-ins if you wish.

# Why Use Web Matrix Instead of Visual Studio .NET?

Before we look in detail at Web Matrix, it's worth exploring the differences between it and Visual Studio .NET. After all, why should Microsoft provide two different development environments for ASP.NET? The answer is that they complement each other – they target different types of development.

Visual Studio .NET is an excellent team development environment, which – when integrated with a source file control system such as Visual SourceSafe – provides for the safe and consistent management of project files when a team of people are working on a project.

One big difference between Web Matrix and Visual Studio .NET is that the latter insists on creating ASP.NET projects using the code-behind technique, rather than inline code. Many traditional ASP developers are used to including the presentation content (such as HTML, text, etc.) in the same file as the ASP code that creates and manages the dynamic interface content. Whether this is a good idea depends on how you (and your team) actually develop applications. If you employ graphical designers to build the visual parts of pages, and then employ other, more technically oriented programmers to build the code, you may prefer to have separate files for these two sections of the interface.

However, you might prefer to include both code and visual content in the same inline page, perhaps to avoid the added complexity of having to compile the code-behind file and then inherit from it in the visual interface page (even though Visual Studio .NET does this for you). Developing in this way, until Web Matrix appeared, meant going back to the pre-ASP.NET approach of using a simple text editor (such as Notepad) or some other third-party tool.

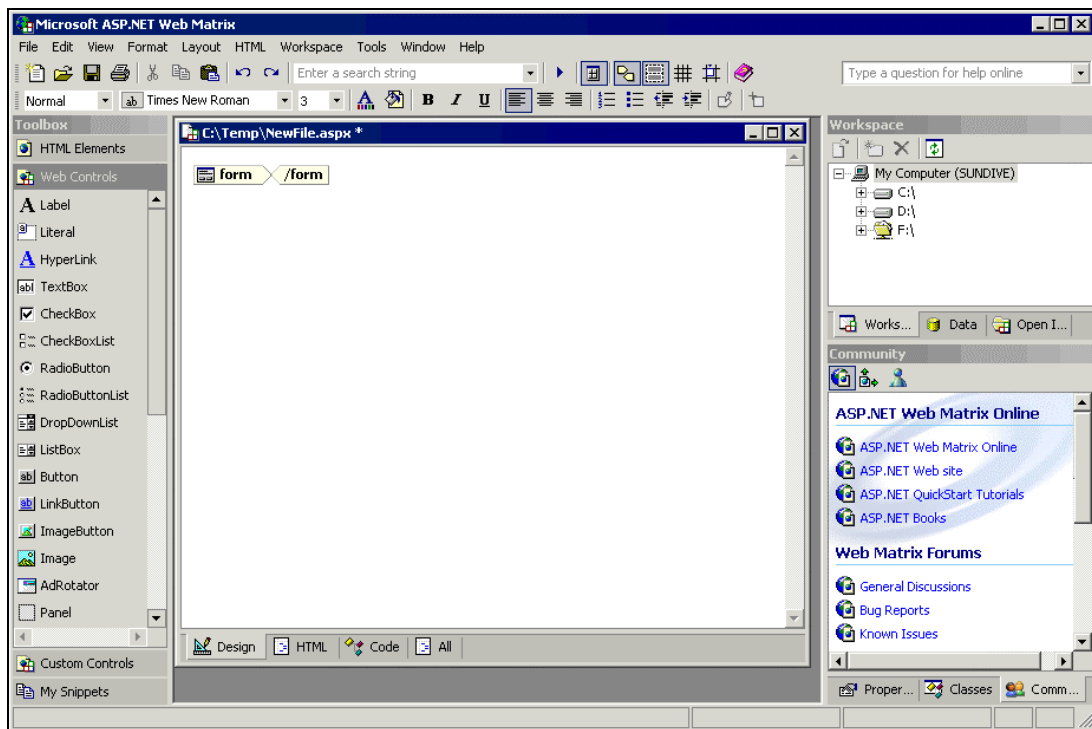
In summary, the differences between Web Matrix and Visual Studio .NET are:

- ❑ **Project-based Solutions** – Visual Studio .NET has the concept of a **project**, to which you can add various types of file and resource. Web Matrix does not use a project-based approach; instead it treats each file as a separate item.
- ❑ **ASP.NET Page Structure** – Web Matrix creates ASP.NET pages using the inline approach, rather than the code-behind approach of Visual Studio .NET.
- ❑ **User Interface** – Web Matrix is light (the installation file is only around 1MB in size), thin, and fast. However, it does not provide the entire set of user interface goodies that are included in Visual Studio .NET. For example, Web Matrix does not provide statement completion, lists of object members, or pop-up tips in the edit window.
- ❑ **Compilation of Class Files** – Unlike Visual Studio .NET, Web Matrix does not automatically compile class files into assemblies. This has to be done from the command line.
- ❑ **.NET Framework Help Files** – Web Matrix does not include reference documentation for the .NET Framework. Instead, it provides a useful, collapsible, folder-based listing of the commonly used classes and their members, along with a full list of all the other namespaces and classes within the .NET Framework Class Library. Web Matrix also ships with a class browser that shows the individual members of any class, and provides a link to the local .NET SDK (if installed) and the online MSDN .NET reference pages.
- ❑ **Community** – Web Matrix is designed to be a community tool, and contains various types of links to the online community site at <http://asp.net/WebMatrix/>, as well as links to newsgroups, list servers, and other sites that provide community support for Web Matrix.
- ❑ **Cost** – Web Matrix is free!

In the words of the Web Matrix development team at Microsoft, Web Matrix is "a viral product designed to support a community approach to ASP.NET development, while being fun to use as well". It aims to focus on those tasks that fulfill 80% of the requirements for building ASP.NET applications.

## A Road Map to the Web Matrix IDE

The Web Matrix IDE is designed to provide a familiar interface for developers who have used Visual Studio .NET. It includes the usual menu bars and toolbars across the top, a toolbox down the left, various "project windows" down the right, and a status bar across the bottom. The following screenshot shows the default layout for a new ASP.NET page:

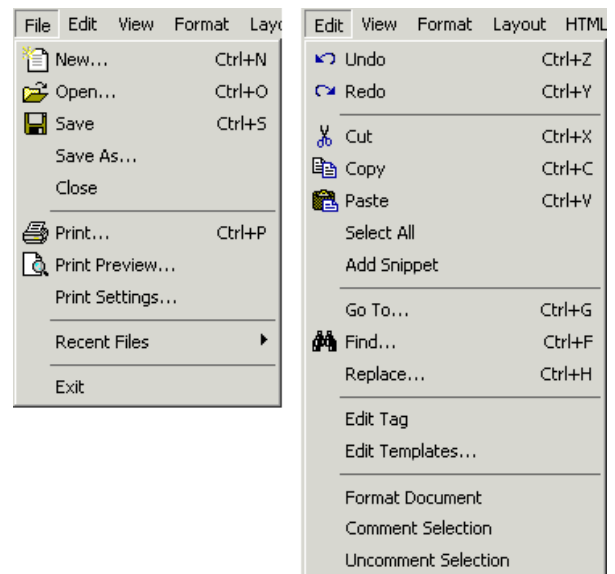


In the current release, the toolbox and project windows are not moveable, though they can be re-sized, and the Toolbox can be hidden. The main work area is a multiple document interface (MDI), so you can open several files at the same time for editing. In the following sections, we'll look in detail at each section of the IDE.

## Menus and Toolbars

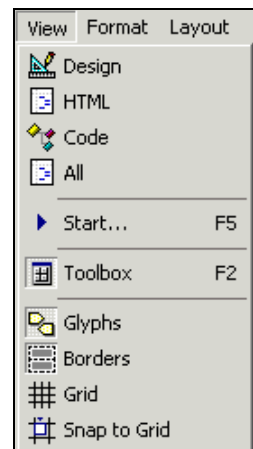
Many of the menu entries and icons on the toolbar are familiar items for any Windows user. The File menu contains commands to create, open, save, close, print, and preview files. The Edit menu contains commands that allow you to undo or re-do your recent edits, to cut, copy, paste, and select text, as well as the usual Find and Replace options and a command to go to a specific line in the page.

The Edit menu also contains commands to add code snippets (something we'll look at when we examine the Toolbox in more detail) and commands to edit the currently selected element when in Design view, or to edit the templates for a DataList control. Again, we'll see more about these features later on. The final group of commands on this menu allows you to format pages, and to comment or un-comment the currently selected text or controls in either HTML or Code view (both of which will be covered in more detail when we look at the edit window later on).

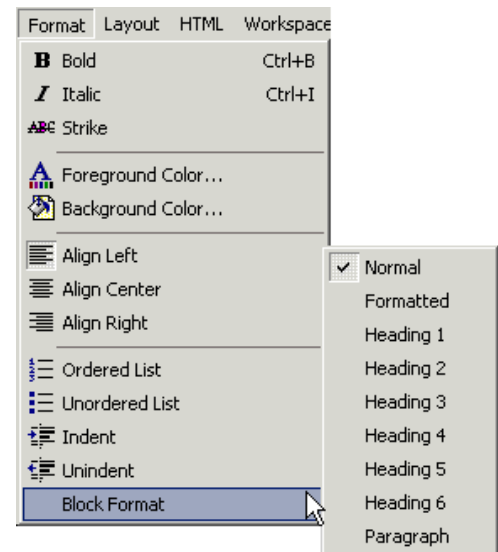




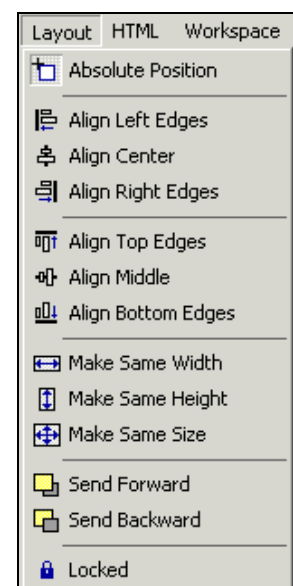
The **View** menu can be used to select a specific view of the current page or file (depending on the type of file). In the earlier screenshot of the IDE itself, you can see tabs at the bottom of the main edit window, which correspond to the four views: **Design**, **HTML**, **Code**, and **All** (all of which are covered in more detail when we look at the edit window later on). The **View** menu also includes commands to start the current ASP.NET page running in your default browser, display or hide the **Toolbox**, toggle the view of **glyphs** (such as the `form` and `/form` items visible in the earlier screenshot of the Web Matrix IDE), show or hide the borders of elements, show or hide the editing grid, and specify if the snap-to-grid feature is on or off.



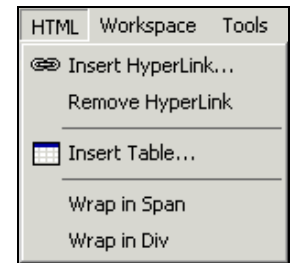
The **Format** menu allows you to specify the appearance of elements in your ASP.NET and HTML pages using the normal combinations of bold, italic, and strikethrough formatting. You can also set the foreground and background colors, control the horizontal alignment, format items as an ordered or unordered list, and indent them. The formatting you specify here is added to the HTML declaration of the elements as a mixture of standard formatting elements (`<b>`, `<i>`, `<font>`, and so on). The final option opens a submenu of block formatting options, which add the relevant "Heading" element, from `<h1>` through to `<h6>`.



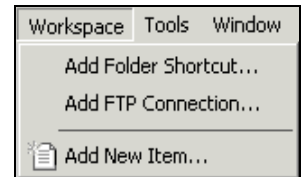
Other, less familiar, top-level menu items are **Layout** and **HTML**. The **Layout** menu contains a command to switch the selected control(s) into **Absolute Position** mode. Web Matrix then adds a `position:absolute` CSS style selector to the element(s), with the appropriate size and position selectors. Then, using the commands on this menu, you can align elements with each other in various ways, make them the same width, height, or size, control their z-order, and lock elements against being moved accidentally.



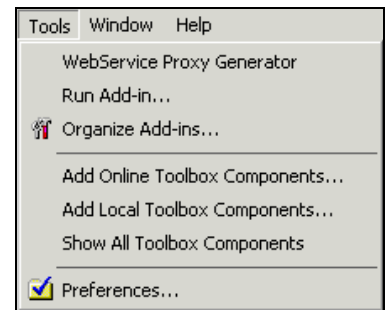
The HTML menu provides features for adding hyperlinks to an element (you get to specify the URL and description), and removing them from an element. It also allows you to insert a fixed-size HTML table, where you can specify all kinds of attributes for it including the number of rows and columns, the borders and colors, the cell spacing, and the cell padding. The commands on the HTML menu can also be used to wrap the currently selected elements in a <span> or <div> element.



The Workspace menu is used to manipulate the Workspace window, which is one of the items displayed on the right-hand side of the IDE (in the section we refer to as the "project windows"). It allows you to add a shortcut to a mapped folder on the network or set up an FTP connection to another server. It can also be used to add other items to the Workspace.

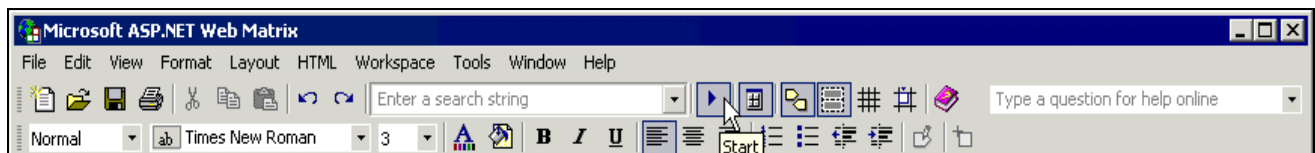


The Tools menu contains commands to run and manage the various add-ins that are provided with Web Matrix, as well as those that you create and those provided by third parties. You can also use it to add components (either stored on your local machine or downloaded from the ASP.NET community site) to the Toolbox – which is displayed on the left-hand side of the IDE – or to reset the Toolbox to show the default list of components. The final command on this menu opens the Preferences dialog, where you can change the way that Web Matrix works to suit your own requirements (we'll look at this in more detail towards the end of this document).



The final two menus, which are not shown here, are Windows and Help. The Windows menu contains the usual Cascade, Tile, and Close All commands, as well as a list of the open windows so that you can quickly switch between them. The Help menu contains the usual commands to work with the local help file, as well as links to related Web sites such as the online .NET documentation and the ASP.NET QuickStart tutorials at MSDN. The Help menu also contains a link that can be used to send feedback to the Web Matrix team at Microsoft, and commands that display information about Web Matrix itself.

Most of the commonly used menu commands are also available from the two toolbars that appear by default in Web Matrix. The following screenshot shows these toolbars, with the Start command (also available from the View menu) highlighted. To the left of this icon is a text box in which you can enter a string to search for in the current file (press *Return* to start the "find" process), and to the right are icons that toggle the display of the Toolbox, the glyphs in the edit window, the element borders and the edit grid, as well as an icon to toggle "snap-to-grid" on and off. The second toolbar contains a collection of common formatting commands. As usual, each toolbar icon displays a descriptive tool-tip when you hover over it with the mouse:



The right-hand end of the upper toolbar contains a combo-box drop-down list in which you can type a question (or relevant keywords) and get help directly from the ASP.NET Web site at <http://www.asp.net/>. This help comes in the form of a list of articles and resources that match your query (again, press *Return* to start the "search" process).

## The Edit Window

Most of the work you will do in Web Matrix involves an instance of the **edit window** (each file you open is displayed in a separate edit window). As you'd expect, the edit window displays the contents of a file, such as an ASP.NET page, a class file, a Web Service, and so on. Depending on what type of file is loaded into the window, you can select one of up to four views of that file:

- ❑ **Design** – which shows the visible appearance of the page (with or without glyphs)
- ❑ **HTML** – which shows the actual HTML and text content of the page, but not any code sections
- ❑ **Code** – which shows just the code in the page without any HTML or other content
- ❑ **All** – which (as you'd guess) shows the complete page, including the page directives and inline code sections

For an ASP.NET page or User Control, you get the full set of options (tabs) shown in the accompanying screenshot. For an HTML page, you only get **Design** and **HTML**. For all other types of file, there are no tabs and the only available view is the content of the file as text.



The items that appear in the Toolbox change depending on the view (we'll explore this a little later when we look at the Toolbox in more detail). In **Design**, **HTML**, and **All** views, you can drag controls from the Toolbox onto the page, and the appropriate HTML is inserted at the point where you drop the control. In **Code** view, you can drag **code builders** onto the page. These are "mini-wizards" that automatically create the code to accomplish specific tasks. Web Matrix comes with a set of extremely useful code builders that create data access methods using SQL statements, and another code builder that creates the code for sending an e-mail message.

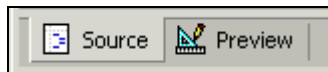
As you edit a file in any one of the views, the other views reflect the changes as well. So you can drag an element onto an ASP.NET page in **Design** view and set its visible appearance, switch to **HTML** view to add attributes to it, and switch to **Code** view to add an event handler for it. Then, switching to **All** view shows the complete page as you'd see it in a normal text editor, with the Page directive, `<script>` block(s), HTML, and all other content visible.

We'll be looking in detail at how the edit window is used in *Part 2*, when we'll start to build ASP.NET pages and other types of files.

### Using Preview Mode

It's possible to change the behavior of Web Matrix's edit window by switching into **Preview Mode**. In this mode Web Matrix doesn't attempt to interpret and display the controls and HTML elements in a page in **Design** view (where they are normally displayed as glyphs). This mode is useful if you have content that was not created within Web Matrix, or which you do not want Web Matrix to interpret and possibly reformat. **Preview Mode** is selected via the Preferences dialog (as we'll see later), and results in only two views (tabs) being available in the edit window:

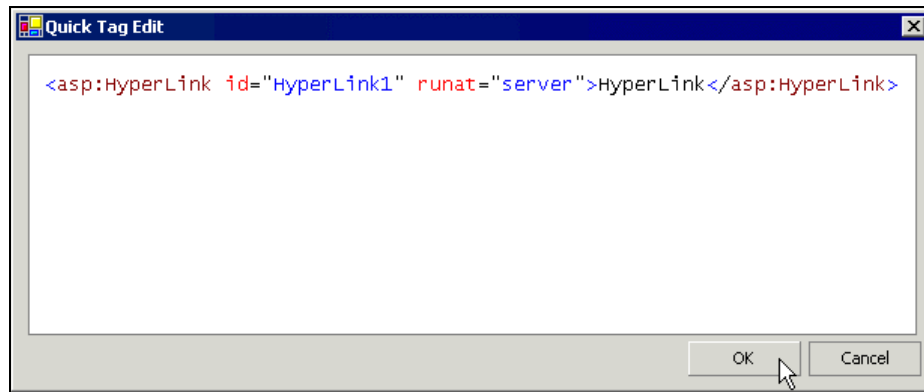
- ❑ **Source** – which shows the complete page, including the page directives and inline code sections (as with the **All** view when in the default editing mode).
- ❑ **Preview** – which shows a rendered preview of the HTML and text content of the page, but without attempting to interpret and reformat it. In appearance it resembles **Design** view, but without the glyphs and without changing the page source.



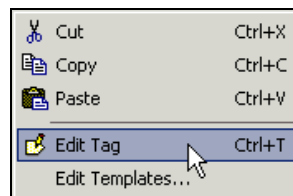
**Preview Mode** only affects ASP.NET pages and User Controls (for which the two views are named **Source** and **Preview**), and HTML pages (for which the two views are named **HTML** and **Preview**).

### Quick Tag Editing

As well as editing the contents of an ASP.NET page or User Control directly in the Edit window, you can use the **Edit Tag** feature on the **Edit** menu to quickly change the attributes or content of an element when the page is displayed in **Design** view. This opens a dialog window that shows the HTML declaration of the currently selected element in the page, which you can then edit as required:



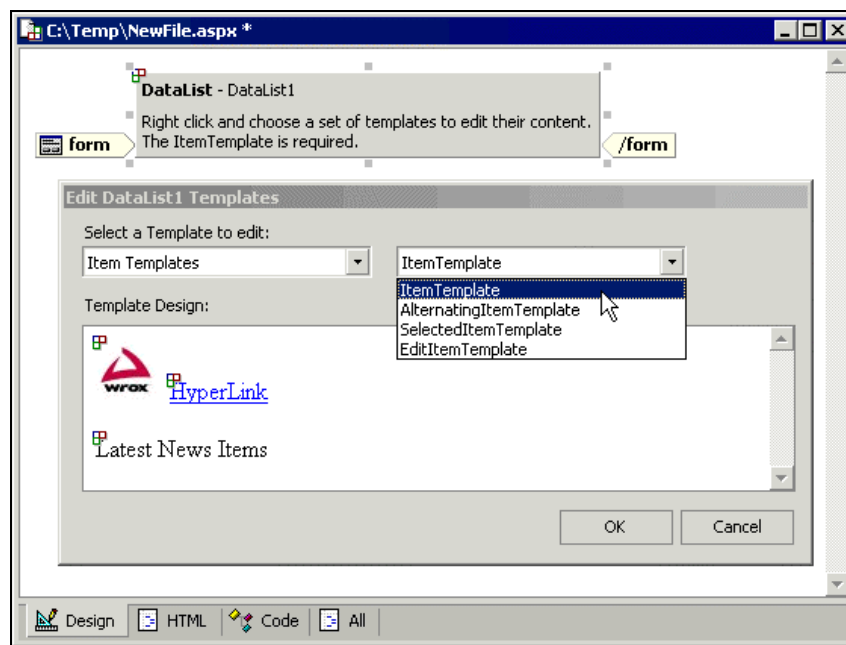
The Quick Tag Edit window can also be opened by right-clicking on an element on the page in Design view, and selecting Edit Tag from the context menu that appears. This menu also contains the Cut, Copy, and Paste commands.



### Editing DataList Templates

When the currently selected control in Design view (for an ASP.NET page or User Control) is a DataList, you can edit the templates for that control using the Edit Templates command on the Edit menu. Alternatively, you can right-click on the DataList control within the page in Design view to open the context menu seen in the previous section, and select Edit Templates. Both actions open the Edit DataList Templates dialog.

Templates define the appearance of the various sections of the control's output at run time, and are divided into three groups: Header and Footer Templates, Item Templates, and Separator Template. Using the two drop-down lists you can specify the template you want to edit and then create the content for the template within the dialog:



The Template Design section of the dialog is a "designer surface", just like the Edit window in Design view. You drag and drop controls onto this surface to create the appearance in exactly the same way as you would in the Edit window.

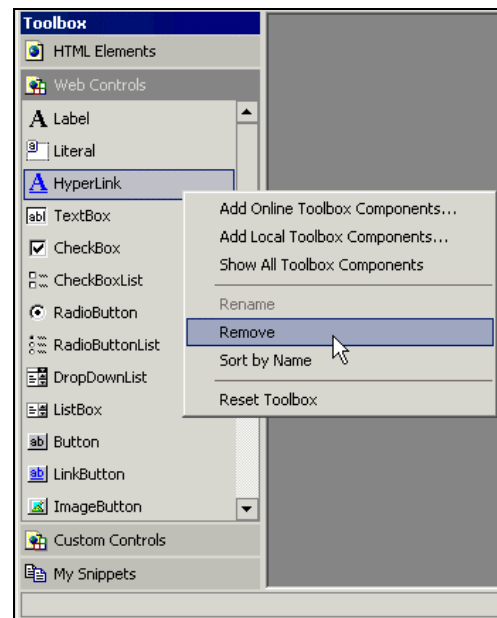
*Templates for a DataGrid control are edited using the Properties window and not through the Edit Templates command. You'll see this demonstrated later in this document. Templates for the Repeater control must be defined manually within the Edit window in HTML view.*

## The Toolbox

The Web Matrix Toolbox looks and works much like those in Visual Studio and other third-party tools. When an ASP.NET page or User Control page is open in Design view, HTML view, or All view within the edit window, the Toolbox lists the various elements and controls that can be added to that page. The available controls are divided into three sections: HTML Elements (server controls from the System.Web.UI.HtmlControls namespace), Web Controls (server controls from the System.Web.UI.WebControls namespace), and Custom Controls (an area where you can add your own controls). To place any of these controls onto the current page, you simply drag them from the Toolbox and drop them into the required position on the page.

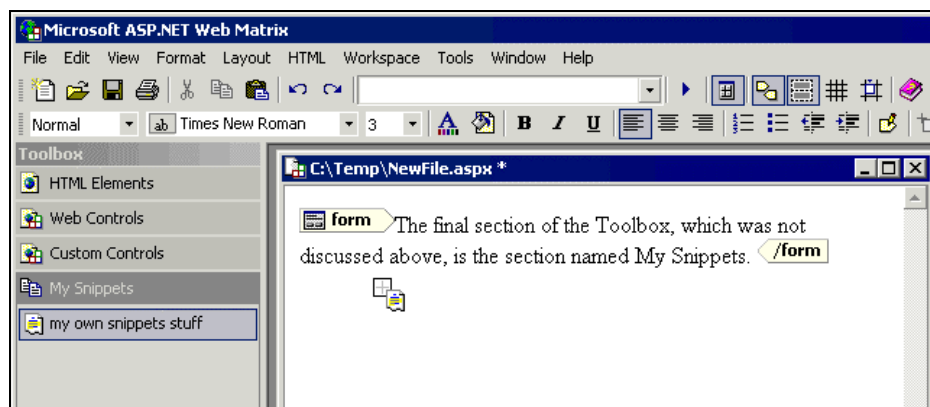
When the current edit window is in Code view or All view, the Toolbox contains a section named Code Builders (which was briefly mentioned earlier). The Code Builders section contains small Wizards that can be used to automate the creation of specific blocks of code in the current page. To start a Code Builder Wizard, you simply drag it from the Toolbox onto the page. If any further information is required, the Wizard will prompt for it.

To manage the contents of the Toolbox, right-click on it (or on one of the controls it contains) and a context menu will appear that provides options to add, remove, and rename controls – as well as options to sort the controls by name and to reset the Toolbox so that it shows its default contents.



## Toolbox Snippets

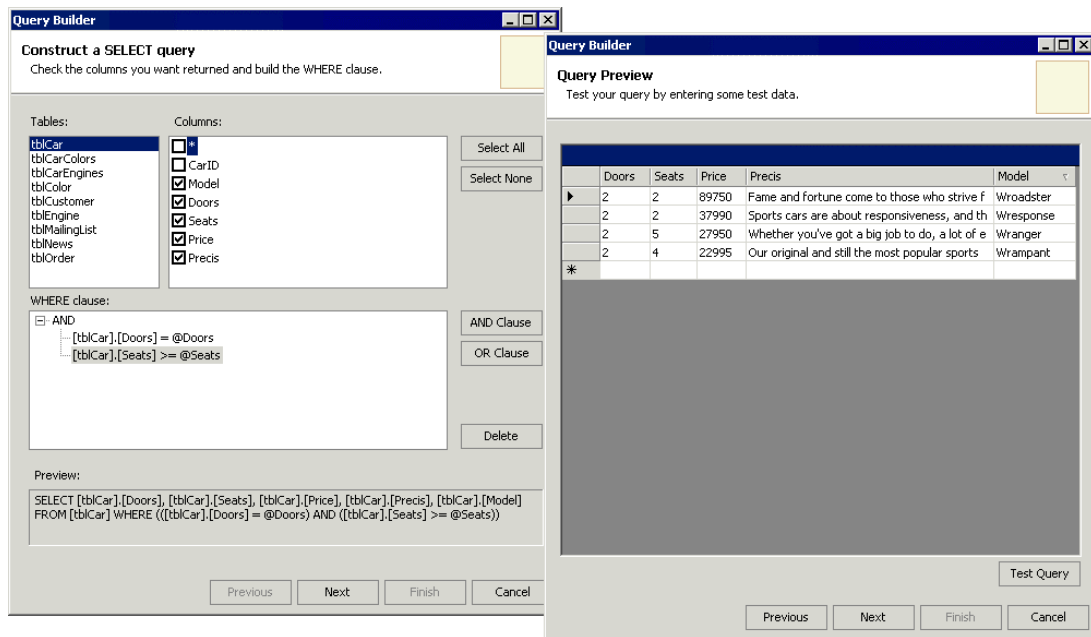
The final section of the Toolbox is named My Snippets (continuing in Microsoft's tradition of creating cute-sounding names). You can create a snippet by highlighting some text or code in the edit window and then simply dragging it onto the Toolbox. Part of the first line of the text is displayed in the Toolbox (although you can rename it as shown in the next screenshot), and the complete snippet appears in a tool-tip when the mouse hovers over it. This snippet can then be inserted into any other page or file by dragging it from the Toolbox:



If you right-click on the My Snippets section of the Toolbox, or on an existing snippet, a context menu that allows you to rename or remove snippets appears. You can also export the snippets to an XML file as well as importing snippets as XML. This is an extremely quick and useful way to share code with others – including the ASP.NET Web Matrix Community. You'll find more about this community later on.

### Toolbox Code Builders

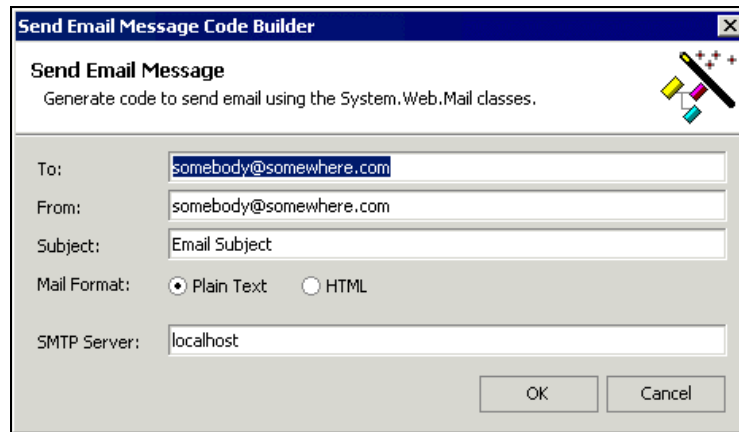
Let's look at an example of the Code Builders that are provided with Web Matrix. The following screenshot shows the results of dragging the SELECT Data Method Code Builder onto an ASP.NET page. After displaying the dialog in which you specify which database to connect to, Query Builder opens a dialog in which you can create the required SQL statement. Once this SQL statement is complete, Query Builder provides a dialog in which you can test the statement by entering values for the WHERE clause parameters, and then view the results:



Once you are satisfied with the SQL statement, Query Builder asks you to specify the name for the data access method it will create, and whether the method will use a DataSet or a DataReader to access the data store. Choosing a DataSet creates code along the lines of the following. The code is inserted into the current page automatically. Note that the WHERE clause contains type-safe arguments. This is achieved through the use of typed parameters in the function:

```
Function MyQueryMethod(ByVal doors As Short, ByVal seats As Short) As System.Data.DataSet
    Dim connectionString As String = "server='localhost'; user id='userid'; " _
        & "password='password'; Database='WroxCars'"
    Dim sqlConnection As System.Data.SqlClient.SqlConnection _
        = New System.Data.SqlClient.SqlConnection(connectionString)
    Dim queryString As String = "SELECT [tblCar].[Doors], [tblCar].[Seats], " _
        & "[tblCar].[Price], [tblCar].[Precis], [tblCar].[Model] " _
        & "FROM [tblCar] WHERE (([tblCar].[Doors] = @Doors) " _
        & "AND ([tblCar].[Seats] >= @Seats))"
    Dim sqlCommand As System.Data.SqlClient.SqlCommand _
        = New System.Data.SqlClient.SqlCommand(queryString, sqlConnection)
    sqlCommand.Parameters.Add("@Doors", System.Data.SqlDbType.SmallInt).Value = doors
    sqlCommand.Parameters.Add("@Seats", System.Data.SqlDbType.SmallInt).Value = seats
    Dim dataAdapter As System.Data.SqlClient.SqlDataAdapter _
        = New System.Data.SqlClient.SqlDataAdapter(sqlCommand)
    Dim dataSet As System.Data.DataSet = New System.Data.DataSet
    dataAdapter.Fill(dataSet)
    Return dataSet
End Function
```

The Send Email Code Builder item can also be dragged onto a page, and in this case the dialog shown in the following screenshot appears. In this dialog you can specify the To and From addresses, the Subject, the mail format, and the SMTP Server to use to send the message:



The following code is then created. All that is left to do is to set the text of the message using the `mailMessage.Body` property:

```
' Build a MailMessage
Dim mailMessage As System.Web.Mail.MailMessage = New System.Web.Mail.MailMessage
mailMessage.From = "somebody@somewhere.com"
mailMessage.To = "somebody@somewhere.com"
mailMessage.Subject = "Email Subject"
mailMessage.BodyFormat = System.Web.Mail.MailFormat.Text

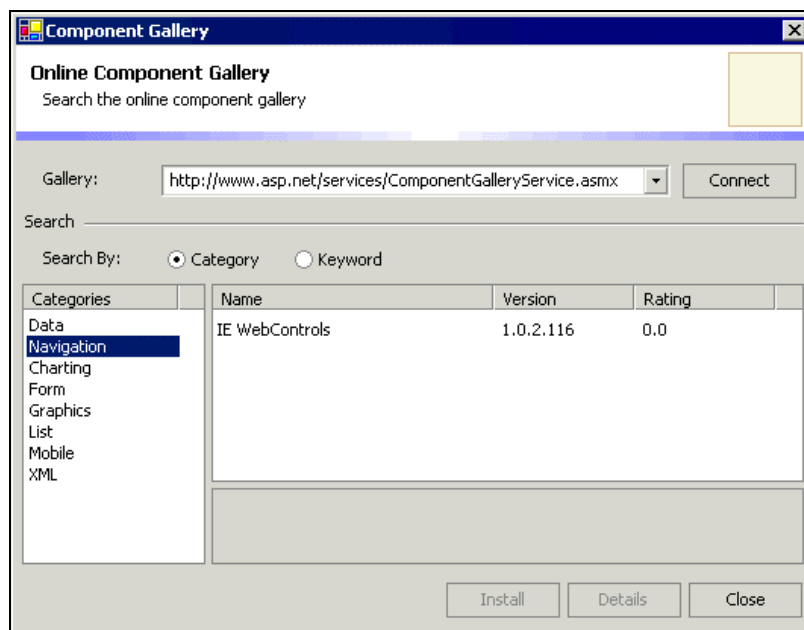
' TODO: Set the mailMessage.Body property

System.Web.Mail.SmtpMail.SmtpServer = "localhost"
System.Web.Mail.SmtpMail.Send(mailMessage)
```

### Adding Controls to the Toolbox

You can add any type of .NET control to the Toolbox – the controls from the Microsoft Mobile Internet Toolkit (MIT), the Internet Explorer Web Controls, or controls that you install from the Web Matrix community Web site. To install controls into the Custom Controls section of the Toolbox, select Add Online Toolbox Components or Add Local Toolbox Components from the main Tools menu, or right-click on the Toolbox itself.

If you select Add Online Toolbox Components, Web Matrix connects to the community Web site and shows the controls that are available for download. In the following screenshot we've selected the DHTML Controls section, and you can see that it contains the IE Web Controls (note that the range of Online Component Gallery controls displayed here will expand and change regularly over time):

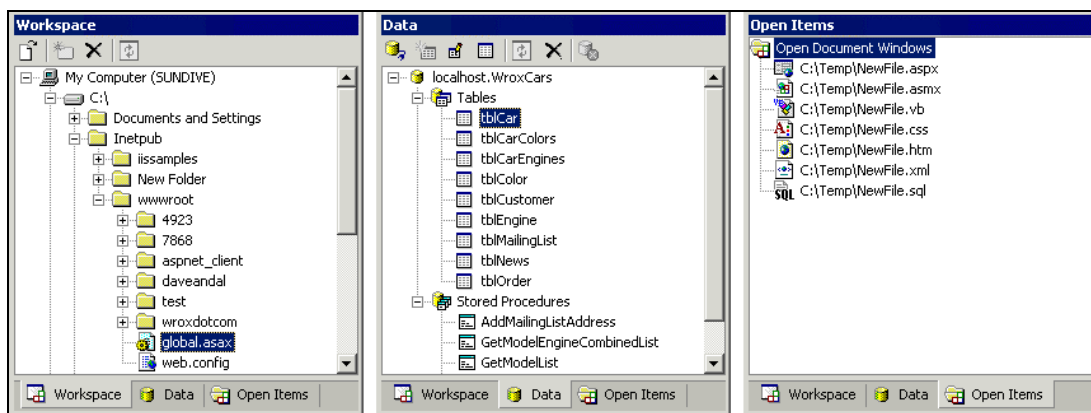


If you select Add Local Toolbox Components, a dialog opens that allows you to select an assembly from the %WINDOWS%\Microsoft.Net\Framework\version\ folder on your system by default, or from any other folder that contains a suitable Component Library DLL.

## The "Project" Window (Upper Half)

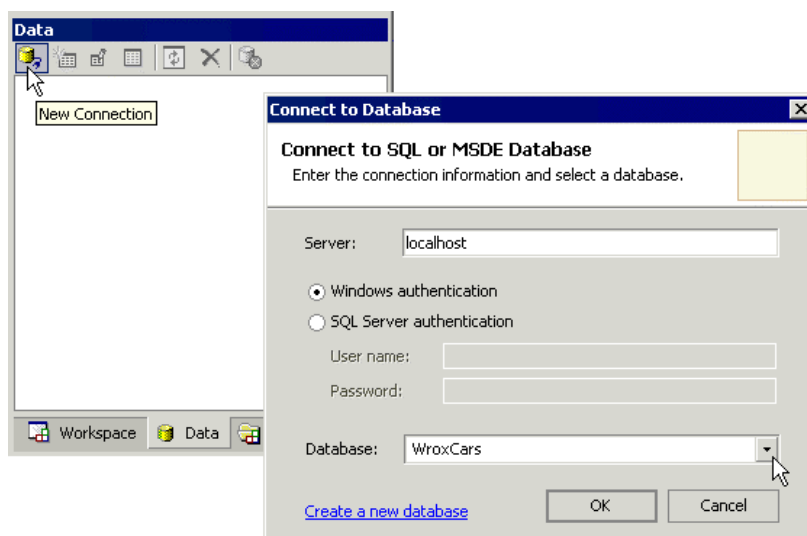
Down the right-hand side of the screen are two windows that together make up what we refer to here as the "project window". The upper half provides three options:

- ❑ **Workspace** – which shows the drives and files on your local machine and, by default, any mapped folders. You can also add shortcuts to other local or network folders, or to other servers via FTP using the commands on the Workspace menu. Double-clicking a file here opens it in a new editing window. By right-clicking in the Workspace window you can add a shortcut, create a new folder, or create a new file based on one of the standard types available in the New File dialog. We'll look at the available file types in *Types of Project and Wizards* later on.
- ❑ **Data** – which allows you to connect to a SQL Server or MSDE database and then view, create, and edit tables and stored procedures within that database. The following screenshot shows a connection to a database named WroxCars, and you can see the tables within this database displayed.
- ❑ **Open Items** – which lists the edit windows that are currently open within the IDE. You can simply click on one to bring it to the front for editing.



## Using the Data Window

To connect to a data source click the icon at the top of the Data window, as shown in the following screenshot, and enter the connection details for the database. You can generally use Windows authentication for the local SQL Server or MSDE database, or if you prefer you can specify the user name and password for SQL Server authentication. Then, select the database from the drop-down list:

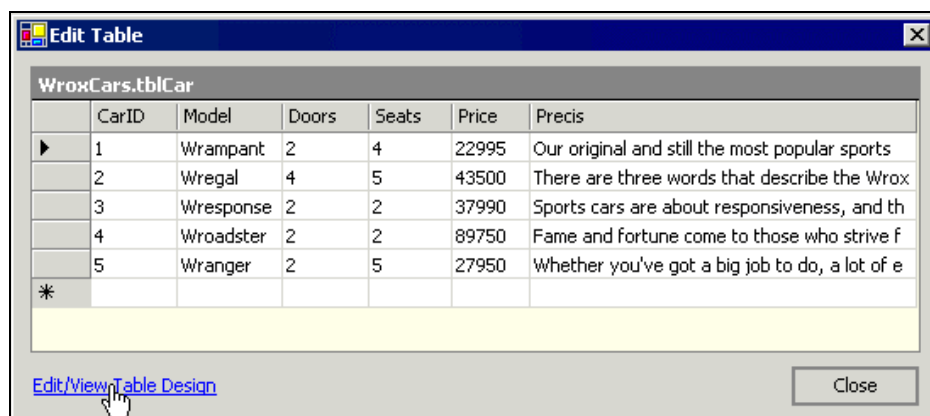




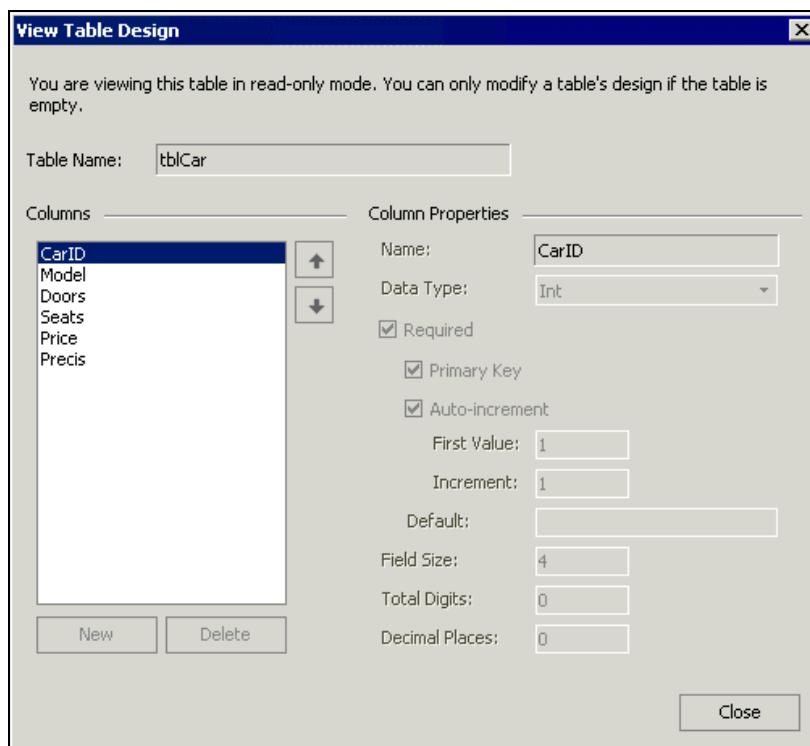
There is also an option to create a new database on the specified server. All you do is enter the name for the new database in the dialog that appears, and the database is created and displayed in the **Data** window.

### Working with a Data Source

Once you have at least one database open in the **Data** window (you can connect to more than one at a time if you wish), the remaining icons at the top of this window are enabled. You can use these icons to work with the database – you can create a new table or stored procedure (using the second icon), or edit existing ones (using the third icon). For example, in the following screenshot, we selected our `tblCars` table and clicked the **Edit** icon. You can see the contents of the table, and the row values can be edited directly. Note, however, that this only works if the table has a primary key defined:



As well as editing the contents of the table, you can also view the table structure by clicking the link at the bottom of the dialog shown in the previous screenshot:



However, you can only edit the table structure if the table is empty. The following screenshot shows a new table that was created by clicking the second (**New Item**) icon in the **Data** window. Three columns have been added to it, setting the properties using the controls in the **Column Properties** section of the dialog:

**Edit Table Design**

When you finish editing, this table will be erased and recreated. Any column/table attributes (including indexes) that are not visible here will be lost.

Table Name:

Columns

- ThePrimaryKey
- AVarCharColumn
- ADefaultColumn

New Delete

Column Properties

Name:

Data Type:

☒ Required

☒ Primary Key

☒ Auto-increment

First Value:

Increment:

Default:

Field Size:

Total Digits:

Decimal Places:

OK Cancel

### Querying a Data Source

The fourth icon in the Data window is used to query a data source. If you select an existing table in the Data window first, Web Matrix creates a simple SQL statement that will query and display the entire contents of that table. You can, of course, edit the SQL statement to perform more specific queries if required:

**Test Query**

**Data Query Tool**

Enter a SQL SELECT Query below:

Results:

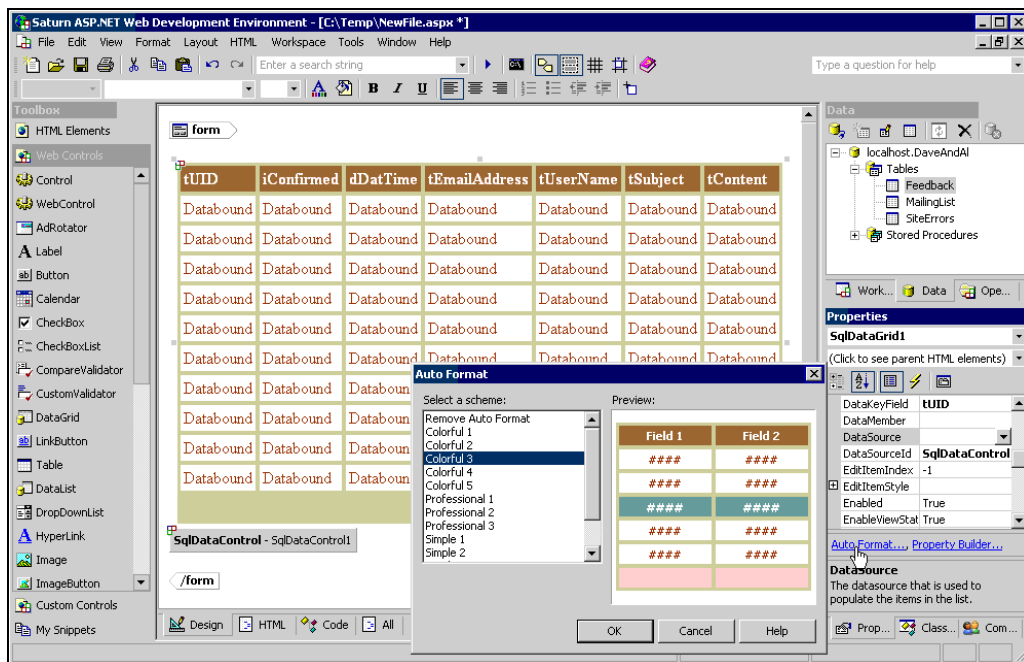
	CarID	Model	Doors	Seats	Price	Precis
►	1	Wrampant	2	4	22995	Our original and still the most popular
	2	Wregal	4	5	43500	There are three words that describe th
	3	Wresponse	2	2	37990	Sports cars are about responsiveness,
	4	Wroadster	2	2	89750	Fame and fortune come to those who
	5	Wranger	2	5	27950	Whether you've got a big job to do, a l

Test Query Close

The other three icons at the top of the Data window can be used to refresh the contents of the window, delete a table or stored procedure from a database (or even delete an entire database), and disconnect a database from the Data window. This means that the Data window provides a complete environment for working with databases, which is especially useful if you are connecting to MSDE (which, unlike the full version of SQL Server, provides no user interface). Just take care not to select the Delete icon when you only want to disconnect!

## Data Tables, Data Controls, and Grid Controls

Data access is at the heart of many Web sites and Web applications, and Web Matrix makes it easy to perform common tasks that interact with a database. For example, you can drag a table from the list in the Data window and drop it onto a page – whereupon Web Matrix automatically creates a DataGrid-based page that displays the data from that table:



If you then switch to Code view, you can see the code that Web Matrix has created within the page (the following is an abridged version):

```
<wmx:SqlDataSourceControl id="SqlDataSourceControl1" runat="server"
    UpdateCommand="UPDATE [tblCar] SET [Model]=@Model,[Doors]=@Doors,[Seats]=@Seats,
        [Price]=@Price,[Precis]=@Precis WHERE [CarID]=@CarID"
    SelectCommand="SELECT * FROM [tblCar]" AutoGenerateUpdateCommand="False"
    ConnectionString="server='localhost'; trusted_connection=true; Database='WroxCars'"
    DeleteCommand="" >
</wmx:SqlDataSourceControl>
<wmx:MxDataGrid id="MxDataGrid1" runat="server" DataSourceControlID="SqlDataSourceControl1"
    BorderColor="#CCCCCC" AllowSorting="True" AutoGenerateFields="False" DataMember="tblCar"
    AllowPaging="True" BackColor="White" CellPadding="3" DataKeyField="CarID"
    BorderWidth="1px" BorderStyle="None" >
    <PagerStyle horizontalalign="Center" forecolor="#000066" backcolor="White"
        mode="NumericPages" ></PagerStyle>
    <FooterStyle forecolor="#000066" backcolor="White" ></FooterStyle>
    <SelectedItemStyle font-bold="True" forecolor="White"
        backcolor="#669999" ></SelectedItemStyle>
    <ItemStyle forecolor="#000066" ></ItemStyle>
    <Fields>
        <wmx:BoundField DataField="CarID" SortExpression="CarID"
            HeaderText="CarID" ></wmx:BoundField>
        <wmx:BoundField DataField="Model" SortExpression="Model"
            HeaderText="Model" ></wmx:BoundField>
        <wmx:BoundField DataField="Doors" SortExpression="Doors"
            HeaderText="Doors" ></wmx:BoundField>
        <wmx:BoundField DataField="Seats" SortExpression="Seats"
            HeaderText="Seats" ></wmx:BoundField>
        <wmx:BoundField DataField="Price" SortExpression="Price"
            HeaderText="Price" ></wmx:BoundField>
        <wmx:BoundField DataField="Precis" SortExpression="Precis"
            HeaderText="Precis" ></wmx:BoundField>
    </Fields>
    <HeaderStyle font-bold="True" forecolor="White" backcolor="#006699" ></HeaderStyle>
</wmx:MxDataGrid>
```

The code uses several new server controls that are specially designed for use in Web Matrix – `MxDataGrid` and the associated `BoundField` elements that create the visible part of the display, as well as `SqlDataSourceControl`, which connects to the data source and exposes the data. We'll look at these in more depth later on.

## The "Project" Window (Lower Half)

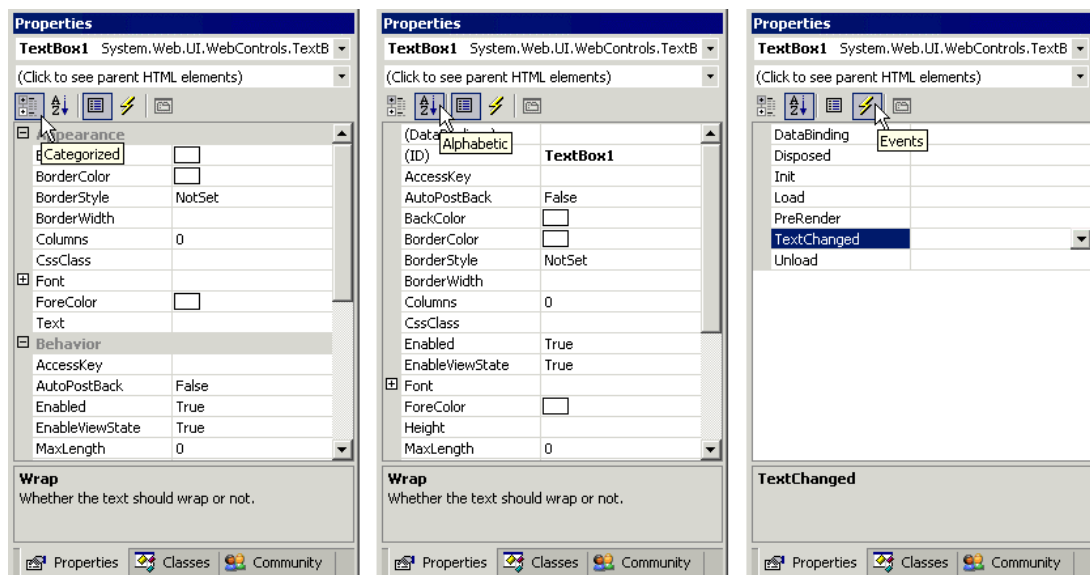
The lower section of the "project" window also provides three main features:

- ❑ **Properties (and Events)** – which allows you to view and edit the properties for the item currently selected in the edit window. For an ASP.NET page or a User Control in Design view, you can select an element (or even the page itself by selecting the `<body>` element) and access its properties. In all other views, the Properties window shows the ASP.NET-related properties for the document itself. For all other types of file, the Properties window simply displays the path to the document.
- ❑ **Classes** – which provides a list of all the .NET Framework namespaces (displayed either by type or name) that are commonly used in ASP.NET projects. Each namespace can be expanded to show the classes it contains, together with the base class and interfaces that it implements. Double-clicking on any entry in this list opens the Web Matrix Class Browser, which shows details of that item.
- ❑ **Community** – which provides links to useful resources, newsgroups, list servers, and other sites that provide support and reference materials for Web Matrix.

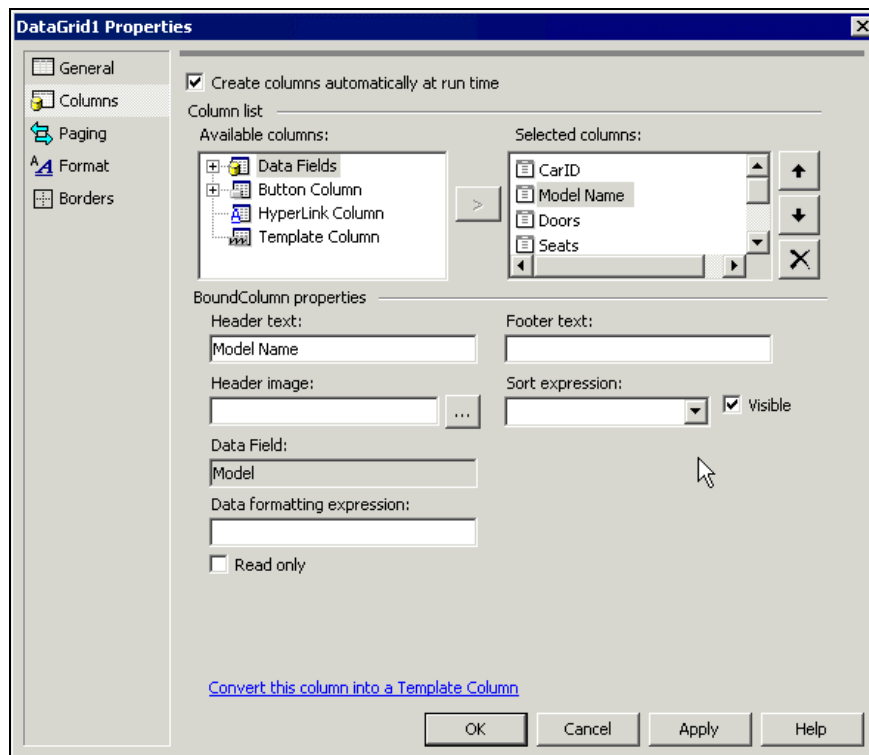
## The Properties Window

The Properties window provides three views of the properties and events for an element or other object. The default view is Categorized (as shown in the first of the following three screenshots), which displays the properties within expandable categories such as Appearance, Data, and Layout. The icons in the Properties window allow you to switch between this view and Alphabetic view, which is useful if you are familiar with the property names for the selected object. In both cases, property values can be set by typing in a new value, selecting it from a list of predefined values (such as `True` or `False`), or through a "picker" (for example, for selecting a color) or some other dialog.

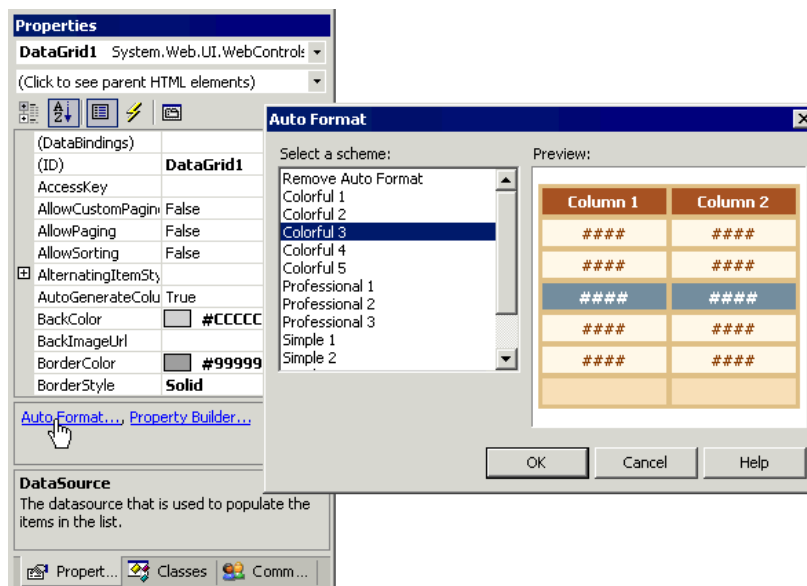
There is also an Events icon in the Properties window, which you click to switch the view to show the server-side ASP.NET events that are available for the selected object. Double-clicking on an event name automatically inserts the code for an empty event handler in the page, displays the page in Code view, and positions the cursor ready for you to type the code for the event. You can also use the drop-down list for each event to connect it to an existing event handler in the page. We'll see this process in action in *Part 2*.



Clicking the fifth (and final) icon opens any Property Pages associated with the selected control. For example, with a `DataGrid` control, a dialog that allows the templates for the control to be defined is displayed:



When the currently selected control is a DataGrid or a DataList, the lower section of the Properties window contains two hyperlinks. The second one, Property Builder, opens the "property page" dialog shown in the previous screenshot. The other link, Auto-Format, provides a list of pre-defined colors and styles for the grid or list, and for its content. The style can be applied simply by selecting it in the Auto Format dialog:



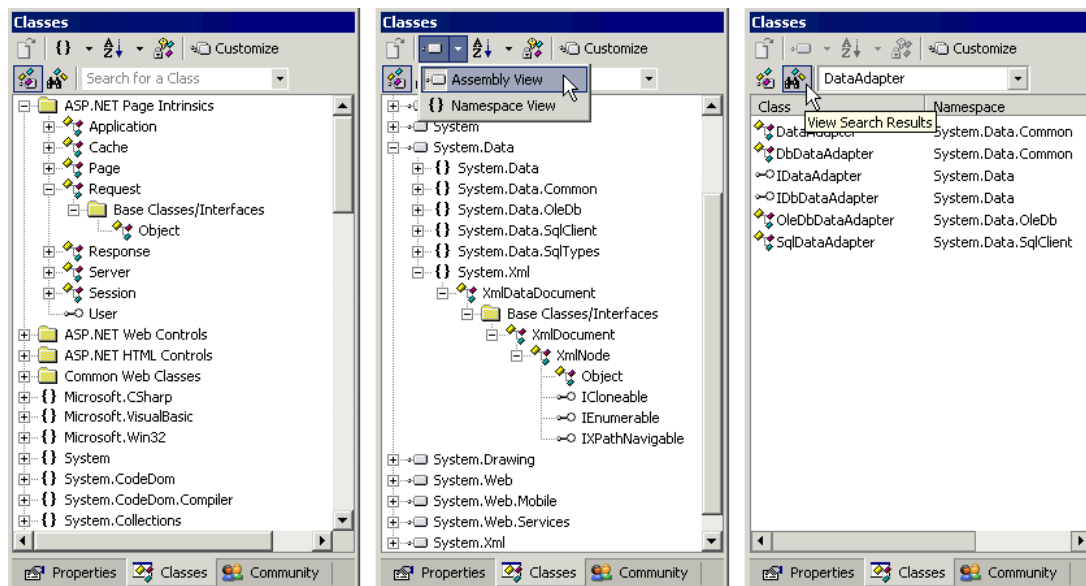
Finally, above the icons in the Properties window, are two useful drop-down lists that make it easy to navigate through the controls on a page. The top one shows the currently selected control, and also lists all the other controls in the page, allowing you to select one directly. The drop-down list below this lists all the elements for which the currently selected element is a child or descendant, and allows you to easily find and select any ancestor or parent (enclosing) element.

## The Classes Window

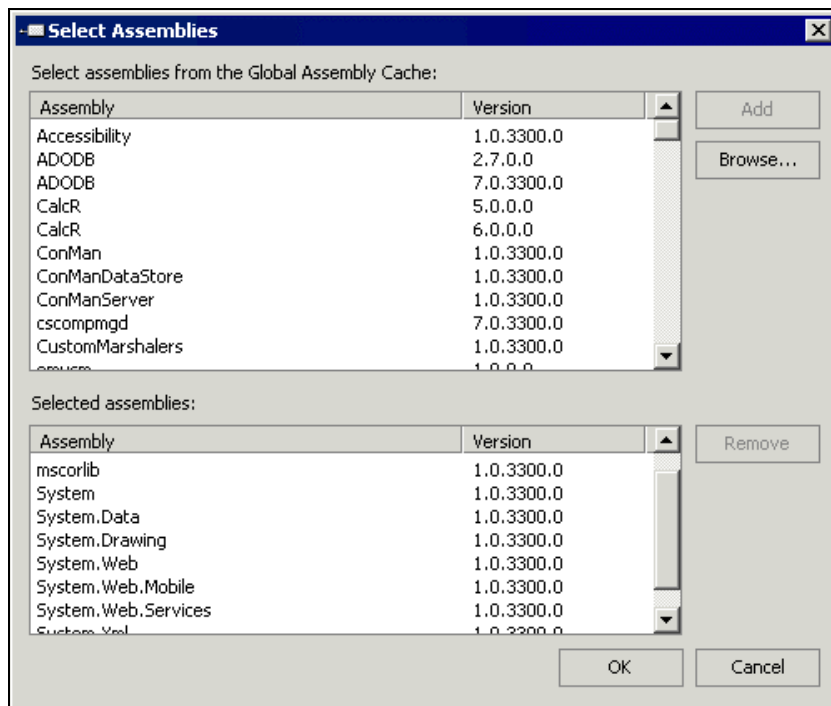
When working with the .NET Framework, information about the multitude of classes that are included in the .NET Framework Class Library is vital. To make it easier, Web Matrix includes features that allow you to access detailed help about any class from within the IDE. The **Classes** window, by default, shows four folders that contain the ASP.NET Page Intrinsic, the range of Web Controls, the range of HTML Controls, and other common classes for Web applications. Below this are the other commonly used classes, listed by namespace.

Other views can be selected; for example you can use the icons at the top of the **Classes** window to display the details in **Assembly view** (where the classes are listed by the assembly DLL that contains them), or the list can be sorted alphabetically, by class type and by visibility (that is, whether they are public or private).

Non-public classes are not shown in the list by default, though the fourth icon in this window can be used to display them in addition to the public classes. The **Classes** window also contains a search text box, in which you can enter a search string. Then, only the classes that contain that string in their name will be displayed (press *Return* to start the search, and then click the **View Search Results** icon to toggle between the search results and the namespace list):

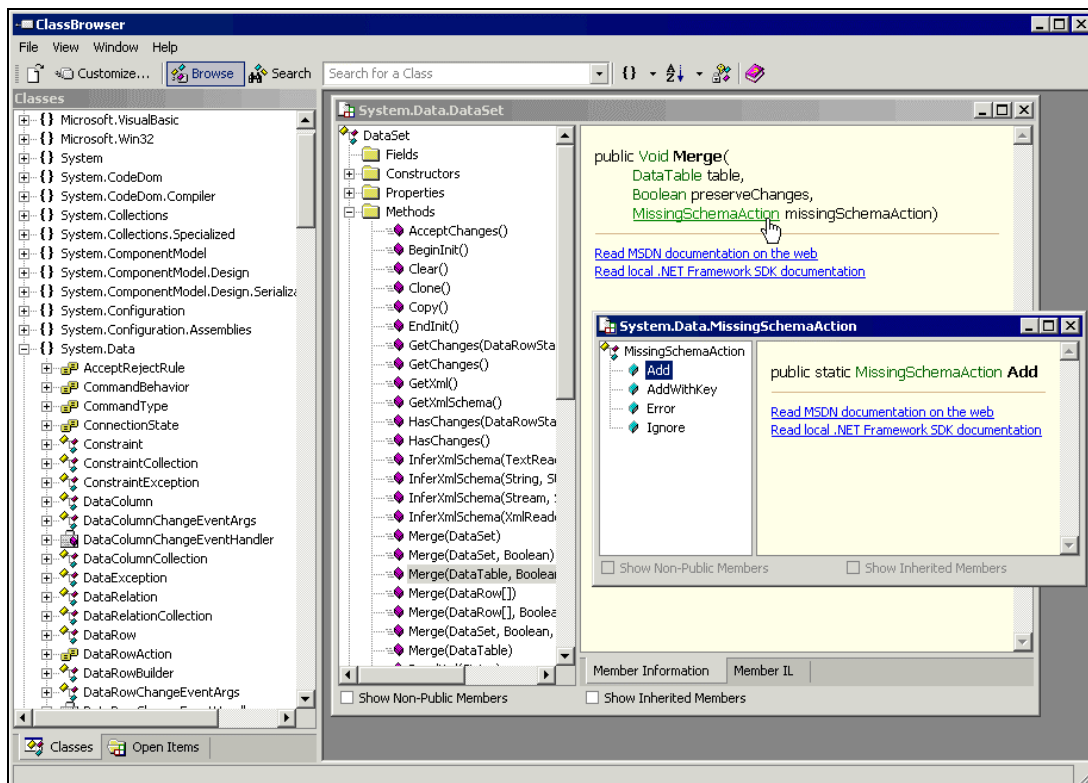


The **Customize** icon in the top right of the **Classes** window is used to modify the list of assemblies that appear in the **Classes** window. The following screenshot shows how to add the **System.EnterpriseServices** assembly to the list in the **Classes** window, which then shows all the namespaces within this assembly:



## The Class Browser

Web Matrix includes a comprehensive tool – the Matrix ClassBrowser – that can be opened independently from the Start menu, or from within Web Matrix by double-clicking on a class displayed in the Classes window. When opened from the Start menu, it looks like the following screenshot. The left-hand window lists the .NET Framework namespaces, and for each one the classes and interfaces implemented within that namespace are listed. Double-clicking on a class opens a list of all the members of that class in a new window in the right-hand area of the Class Browser. Clicking on one of the members then displays details of that member in the right-hand area of this window – including the static fields, constructors, properties, methods, and events:



The **Class Browser** uses reflection to obtain details of the class and its members, so the information is limited to just the member definitions. Each parameter or enumeration listed in the right-hand pane acts as a hyperlink, which when clicked displays information about that object or enumeration in another window. This makes it easy to follow the hierarchy and to get details about the method parameters or property/field value types that are required. There are also links that open the local .NET Framework SDK documentation (if installed) or the online MSDN Library at the relevant page, where more details of the class and its members can be found.

The menu bar and toolbar in the **Class Browser** offer the same set of features as the **Classes** window we discussed earlier. You can change the sort order and organization of the namespaces and classes in the left-hand window, search for classes by name, and show or hide non-public classes. The same **Customize** button, which opens the **Select Assemblies** dialog in which you can add and remove assemblies from the right-hand list, is also present. The **Window** menu can be used to tile or cascade the windows, or just to switch from one to another.

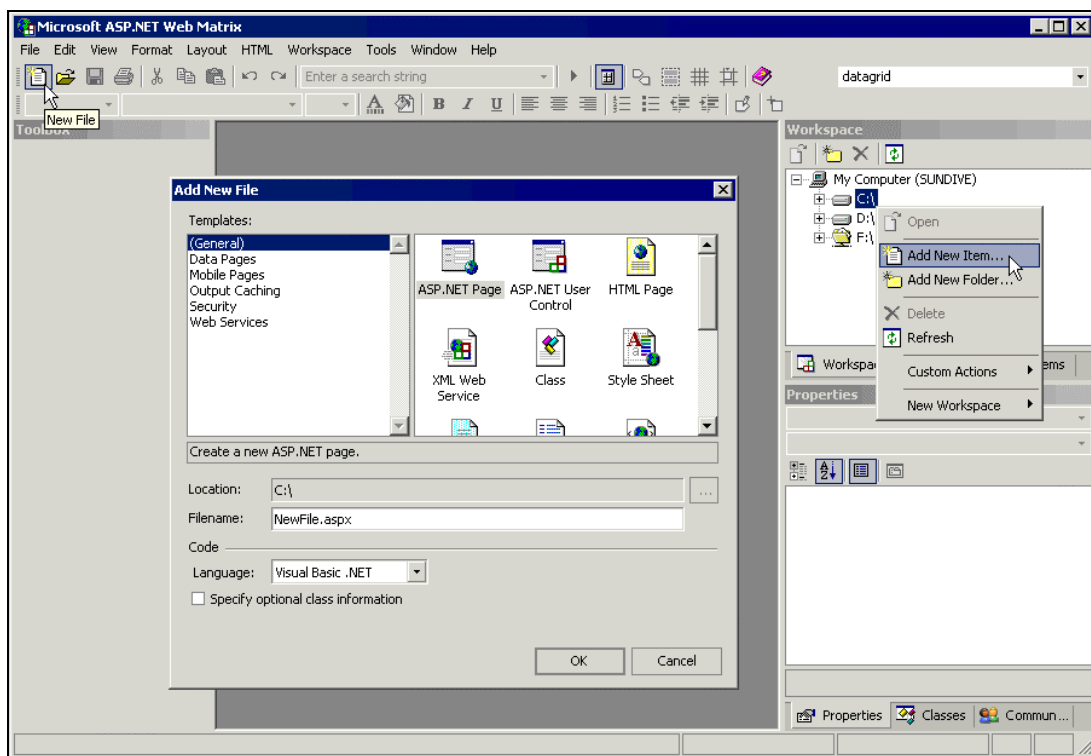
When information about a class is opened from within the Web Matrix IDE (by double-clicking on a class in the **Classes** window) the same right-hand sections of the page shown in the previous screenshot are displayed within the IDE, but the left-hand list of namespaces is not shown.

## Types of Files and Wizards

There are three ways to open a new file in Web Matrix. You can:

- ☐ Select **New** from the **File** menu
- ☐ Click the **New File** icon in the toolbar
- ☐ Right-click on the target folder in the **Workspace** window and select **Add New Item**

The last two of these techniques are shown in the following compound screenshot, together with the **New File** dialog that appears. Notice that the path is that selected in the **Workspace** window:



The **New File** dialog lists the various kinds of file that you can create and edit within the Web Matrix IDE. Each is created from a template stored in folders within the **Templates** subfolder of the **Program Files\Microsoft ASP.NET Web Matrix\version\** folder. Each is described next.



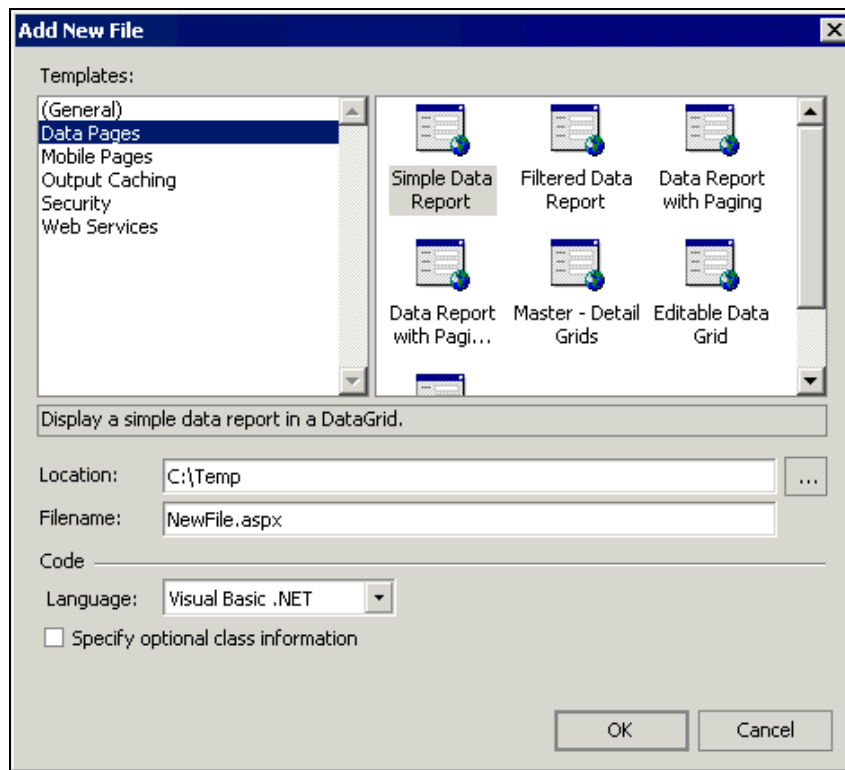
## File Types in the (General) Section

There are 14 different types of file that you can create from the (General) section of the New File dialog. They are:

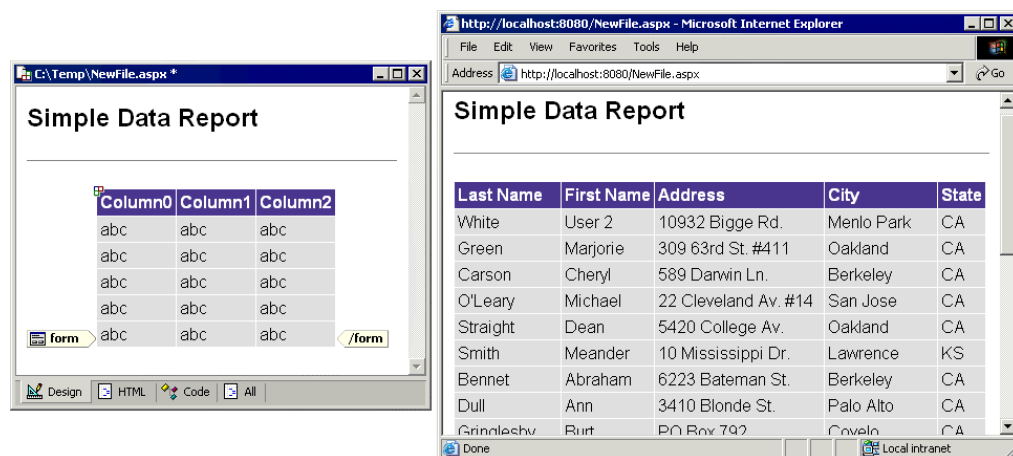
- ☐ **ASP.NET Page** – this creates a file with the extension .aspx. The file contains the @Page directive, the opening and closing <html> tags, an empty <head> section, and a <body> section containing a server-side <form>.
- ☐ **ASP.NET User Control** – this creates a file with the extension .ascx. The file contains just the @Control directive.
- ☐ **HTML Page** – this creates a file with the extension .htm. The file contains the opening and closing <html> tags, plus empty <head> and <body> sections.
- ☐ **XML Web Service** – this creates a file with the extension .asmx. The file contains the @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. You must specify the class name and namespace before you can create this type of file.
- ☐ **Class File** – this creates a file with the extension .vb or .cs (depending on which language you specify), containing an Imports or using statement for the System namespace, the Namespace definition, an outline Class definition, and an empty public Sub or function. You must specify the class name and namespace before you can create this type of file.
- ☐ **Style Sheet** – this creates a file with the extension .css. The file contains just an empty BODY{ } selector definition.
- ☐ **Global.asax** – this creates a file with the extension .asax. The file contains the @Application directive and a <script> section that contains empty event handlers for the Application\_Start, Application\_End, Application\_Error, Session\_Start, and Session\_End events.
- ☐ **Web.Config** – this creates a web.config file containing the <configuration>, <appSettings>, and <system.web> sections. Within the <system.web> section there are <sessionState>, <customErrors>, <authentication>, and <authorization> elements. All the elements are commented-out by default, and contain a description of their usage and the valid values for the common attributes.
- ☐ **XML File** – this creates a file with the extension .xml. The file contains just the <?xml ... ?> processing instruction that defines the version and encoding of the file.
- ☐ **XSL Transform** – this creates an XSLT stylesheet file with the extension .xslt. The file contains the <?xml ... ?> processing instruction and the root <stylesheet> element.
- ☐ **XML Schema** – this creates an XML (XSD) schema file with the extension .xsd. The file contains the <?xml ... ?> processing instruction and the root <xsd:schema> element.
- ☐ **ASP.NET HTTP Handler** – this creates a file with the extension .ashx. The file contains the @WebHandler directive, Imports or using statements for the required System and System.Web namespaces, a Namespace definition, and a public Class definition that implements the IHttpHandler interface. Within the Class definition are the two required member definitions for the ProcessRequest event and the IsReusable property. You must specify the class name and namespace before you can create this type of file.
- ☐ **Text File** – this creates an empty text file with the extension .txt.
- ☐ **SQL Script** – this creates a text file that contains just "/\* New SQL script \*/". The file has an extension of .sql.

## File Types in the Data Pages Section

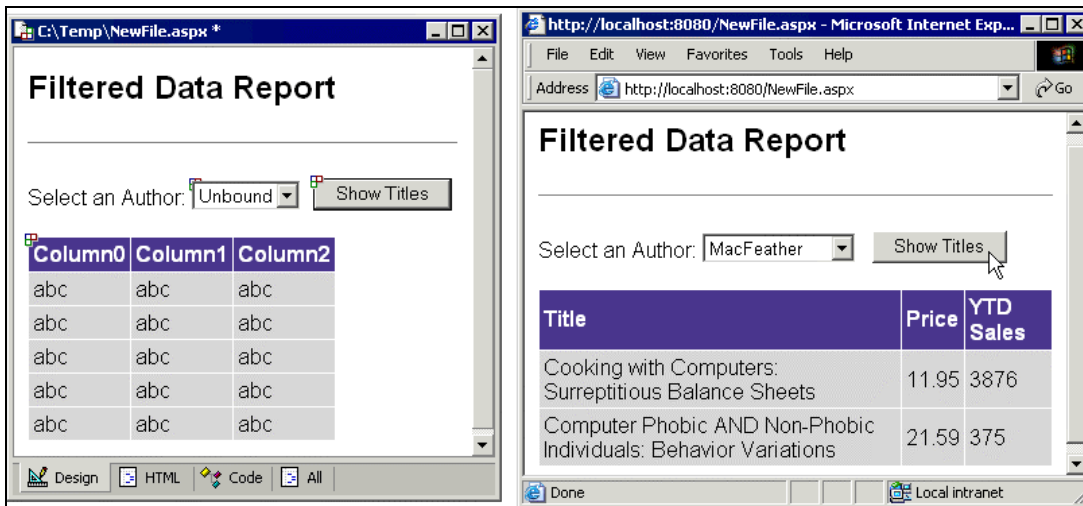
While the file types listed in the (General) section are predominantly empty "outline" files, the file types listed in the other sections are more like Wizards, but without any step-by-step dialogs. The templates that they use to create the new file contain code and (in some cases) ASP.NET server controls in order to implement a working page that you can use as a starting point for developing your own pages. The file types available in the Data Pages section are shown in the following screenshot, and are then described:



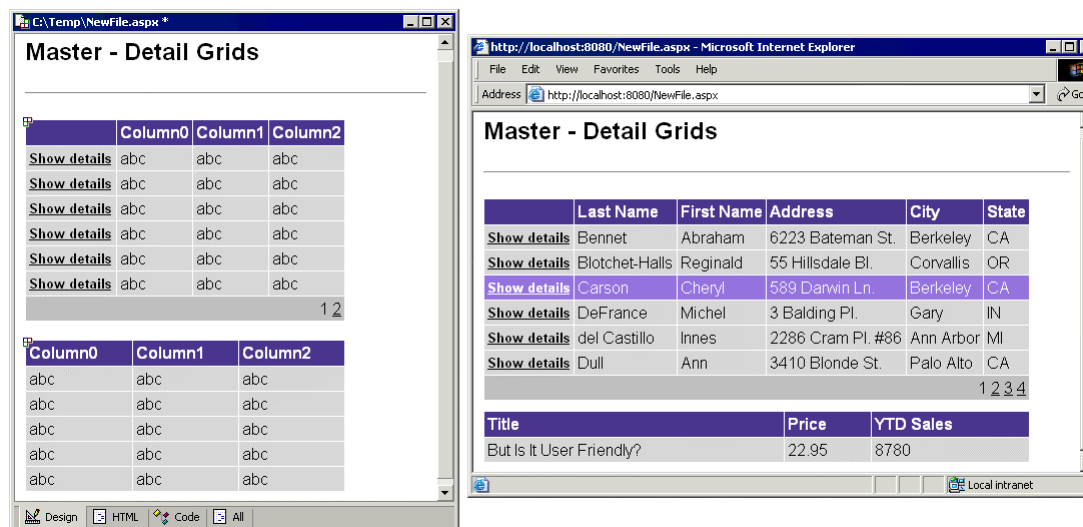
- ❑ **Simple Data Report** – this creates a page that accesses the local SQL Server or MSDE database and displays details from the Authors table of the sample pubs database, using a `DataReader` as the data source for an ASP.NET `DataGrid` control. The following screenshot shows the page in Design view, and when opened in a browser:



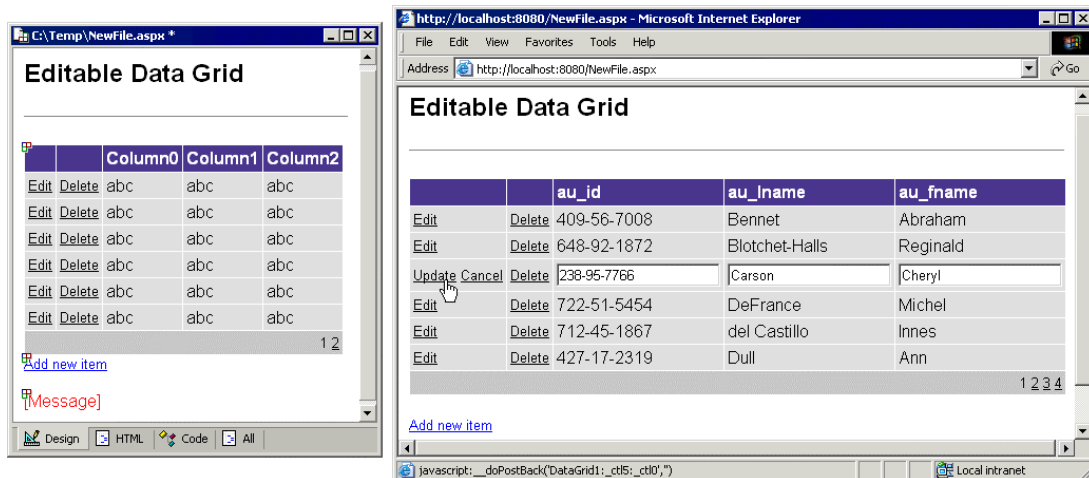
- ❑ **Filtered Data Report** – this creates a similar page to the previous example, but this time containing controls where you can select an author and see a list of their books. A `DataReader` is used when the page is first loaded to fill the drop-down list. Then, when **Show Titles** is clicked, an appropriate SQL statement is constructed and used with a `DataReader` to extract the information from the `titleview` SQL View in the `pubs` database. The following screenshot shows the page in Design view, and when opened in a browser:



- ❑ **Data Report with Paging** – this page fills a DataSet with data from the Authors table and then binds it to an ASP.NET DataGrid control. This time, however, it uses the built-in paging features of the DataGrid to display the results over separate pages, along with links that allow a user to navigate through the pages.
- ❑ **Data Report with Paging and Sorting** – this page extends the techniques developed in the previous type of page by adding a sorting facility. It does this by setting the attributes of the DataGrid control, and adding a simple event handler to respond to the Sort event of the grid.
- ❑ **Master-Details Grids** – this page shows how easy it is to display data from two related tables. Data from the Authors table in the sample pubs database is loaded into a DataSet and displayed in a DataGrid control that has paging enabled, and which contains a ButtonColumn with the text Show details. Clicking on one of these button links causes a DataReader to fetch the matching data from the titleview SQL View in the pubs database and this is displayed in the second DataGrid control. The following screenshot shows the page, both in Design view and when opened in a browser:



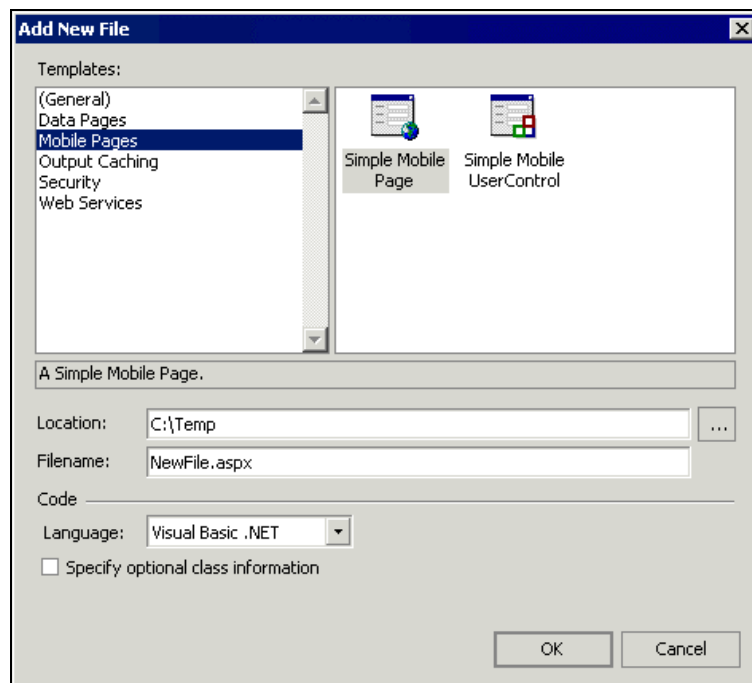
- ❑ **Editable Data Grid** – this example shows the basic technique for editing data using the built-in features of the ASP.NET DataGrid control. A DataSet is filled with data from the Authors table. An EditCommandColumn that implements the Edit/Update/Cancel link buttons and a ButtonColumn that implements the Delete link buttons are then added to this DataGrid. Code in the page reacts to the various events raised by these link buttons; the code executes SQL statements that update the original database table contents. The following screenshot shows the page, in both Design view and when opened in a browser:



- ❑ **Simple Stored Procedure** – this example is similar to the first of the Data Pages we looked at. The only difference is that it calls the stored procedure named `CustOrdersDetail` within the `pubs` database, rather than using a SQL statement stored as a string. The result is returned as a `DataReader`, which is bound to an ASP.NET `DataGrid` control in the page.

## File Types in the Mobile Pages Section

Web Matrix contains templates that allow you to create "mobile" pages and users controls. These pages are based on the classes exposed by the Microsoft Mobile Internet Toolkit (MMIT). The MMIT can be used to create pages that automatically adapt for a range of devices. These pages, and the controls they contain, produce either HTML or WML (Wireless Markup Language) output, tailoring it for the particular device that is accessing the page:



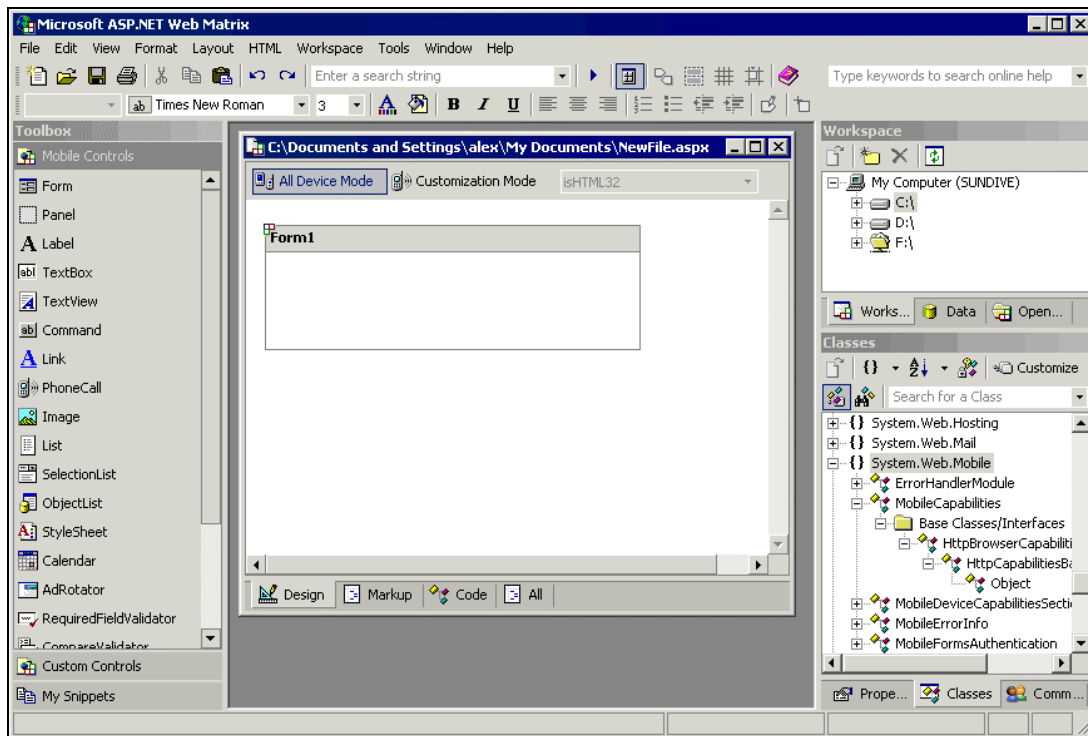
There are two types of file in the Mobile Pages section of the Add New File dialog:

- ❑ **Simple Mobile Page** – this creates a file with the extension `.aspx`. The file contains the `@Page` directive and inherits from the special `MobilePage` class that is implemented within the Microsoft Mobile Internet Toolkit (MMIT). The file also includes the `@Register` directive for the Mobile Controls and an empty server-side `<mobile:Form>`.
- ❑ **Simple Mobile User Control** – this creates a file with the extension `.ascx`. The file contains the `@Control` directive but inherits from the special `MobileUserControl` class implemented within the Microsoft Mobile Internet Toolkit (MMIT). The file also includes the `@Register` directive for the Mobile Controls.

## The Environment for Mobile Pages

When the page currently being edited within Web Matrix is a mobile page, the environment changes to provide the special features required for this type of page. The Toolbar now shows the Mobile Controls section, which contains the controls from the MMIT. These are the only controls that should be used on mobile pages, as the standard HTML and Web Forms controls cannot output the correct content in all circumstances (because they can't produce WML).

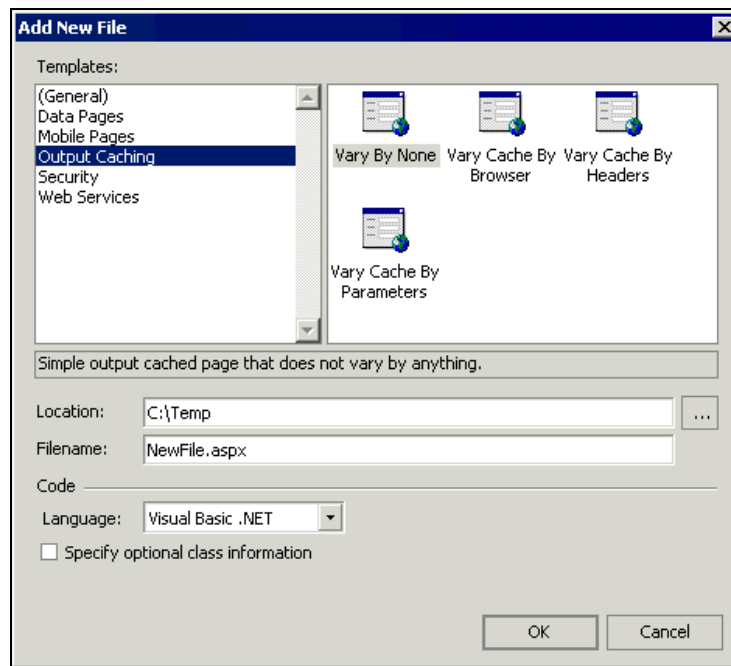
The following screenshot shows that the Edit window changes as well. It gains controls to specify how the page will filter on and react to different devices. In the MMIT, it's possible to set up device filters, so that sections of the output can be modified for specific devices. These controls are used to configure that filtering:



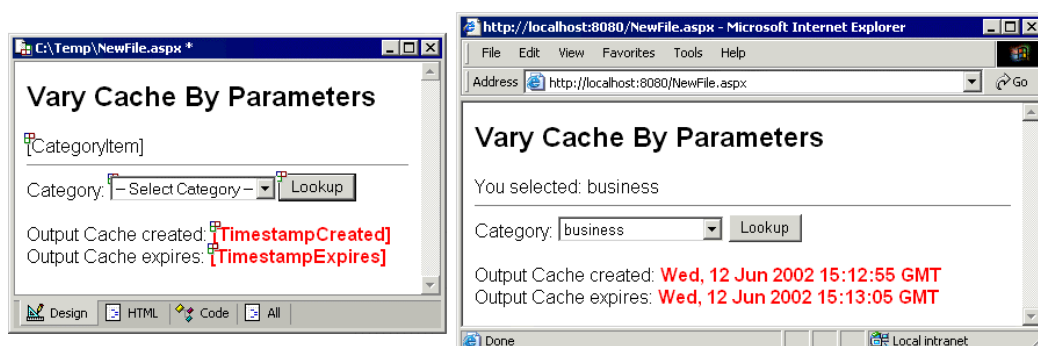
At the bottom of the Edit window, there is a minor change to the tabs for the four different views of the page. Markup replaces HTML on the second tab, as the output from a mobile page may be WML instead of HTML. In addition, the controls and classes from the Mobile Internet Toolkit are included in the default list of classes displayed in the Classes window, and in the separate ClassBrowser tool.

## File Types in the Output Caching Section

The Output Caching section of the New File dialog contains examples of how you can set up pages that use output caching to improve performance, minimize server overhead, and reduce response times. The four available file types are fundamentally similar, and demonstrate how output caching can be configured to automatically detect different features of each request:

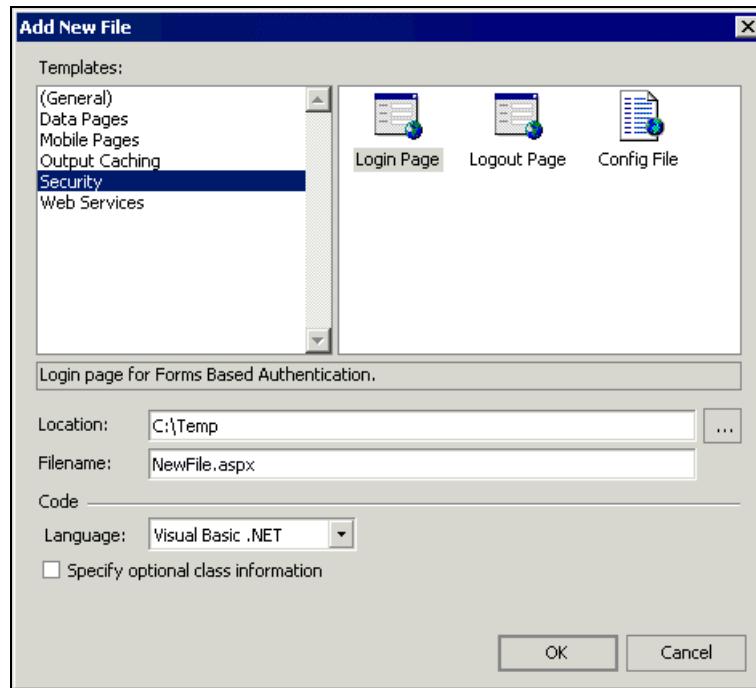


- ❑ **Vary By None** – this demonstrates "total" output caching, where every client is sent the same cached pages until it expires. The example contains an @OutputCache directive that specifies a cache duration of 10 seconds, and contains the attribute VaryByParam="none". Two label controls on the page are set to the current time and the time that the cache expires. By refreshing the page in the browser, you can see the effect of the output caching.
- ❑ **Vary Cache By Browser** – this demonstrates output caching that sends different pages to each type of browser, based on the browser detection carried out by the Request.Browser object. The example contains an @OutputCache directive that specifies a cache duration of 10 seconds, and contains the attributes VaryByParam="none" and VaryByCustom="browser". This time there are three label controls on the page, which are set to the browser name, the current time, and the time that the cache expires. By refreshing the page in the browser, and loading it into different types of browser, you can see the effect of the output caching.
- ❑ **Vary Cache By Headers** – this demonstrates output caching that sends different pages depending on a specific HTTP header sent in the request. This example contains an @OutputCache directive that again specifies a 10 second cache duration, with the attributes VaryByParam="none" and VaryByHeader="Accept-Language". The same page will then only be sent in response to requests where the Accept-Language header is the same. Three label controls on the page are set to the value of the Accept-Language header, the current time, and the time that the cache expires.
- ❑ **Vary Cache By Parameters** – this demonstrates output caching that sends different pages depending on a value sent as a parameter from the client – in this case a value posted from a drop-down list control on a <form>. This example contains an @OutputCache directive that specifies a cache duration of 120 seconds, with the attribute VaryByParam="Category" (the id and name of the drop-down list). Three label controls on the page are set to the value selected in the drop-down list, the current time, and the time that the cache expires. Selecting a different category and clicking the Lookup button causes a page to be created and cached for that category only if one is not already available in the cache. The following screenshot shows this page, in both Design view and when opened in a browser:

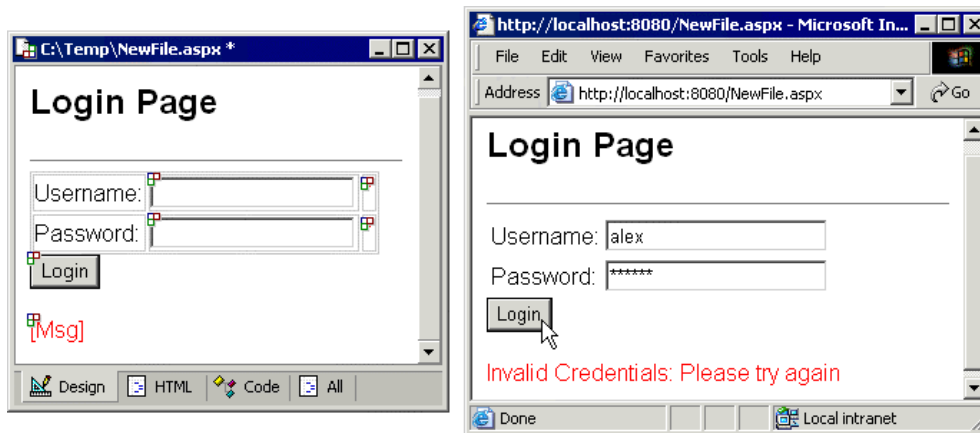


## File Types in the Security Section

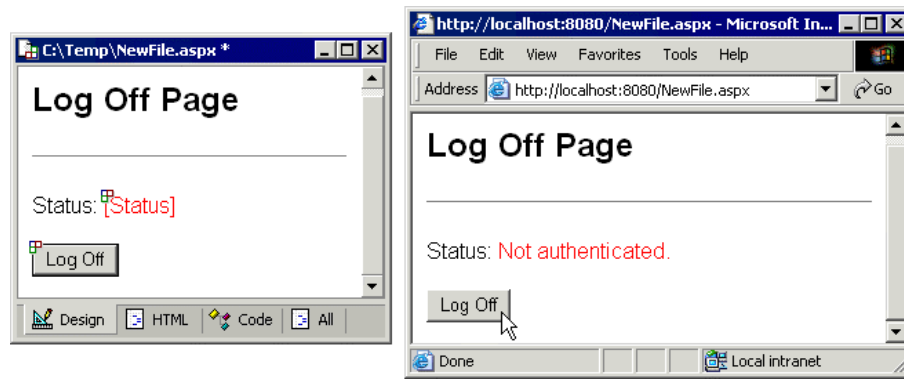
There are three examples in the Security section of the New File dialog, which demonstrate the common techniques for creating authentication (login) pages for a secure section of a Web site:



- ❑ **Login Page** – this creates a standard "log in" page that contains two text boxes 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 screenshot shows this page, both in Design view and when opened in a browser:



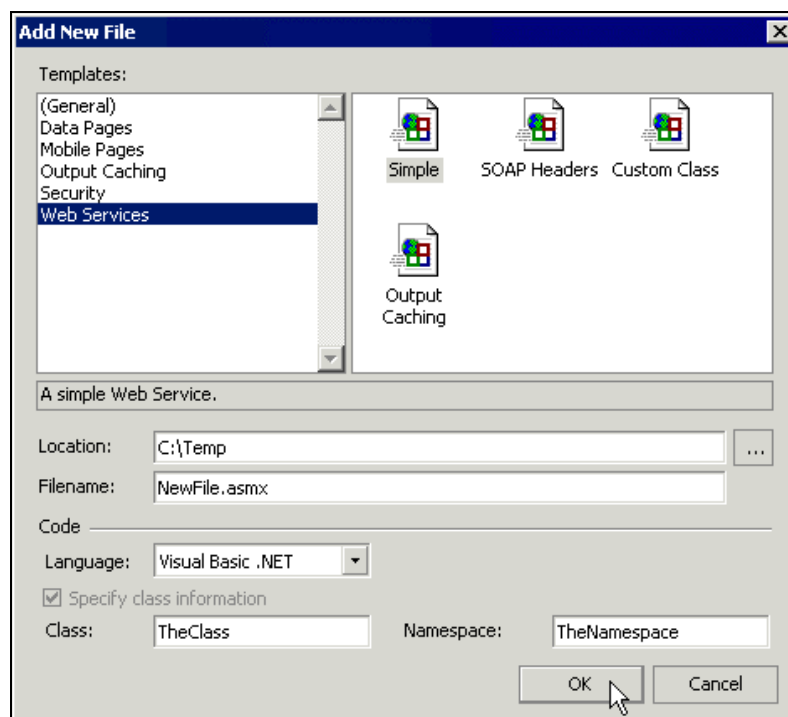
- ❑ **Logout Page** – this creates the corresponding "log off" page, with a Status label and a single Log Off button. The label shows the username of the currently logged-in user where available. Clicking the button calls the SignOut method and displays a message indicating that the user is no longer authenticated. The following screenshot shows 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

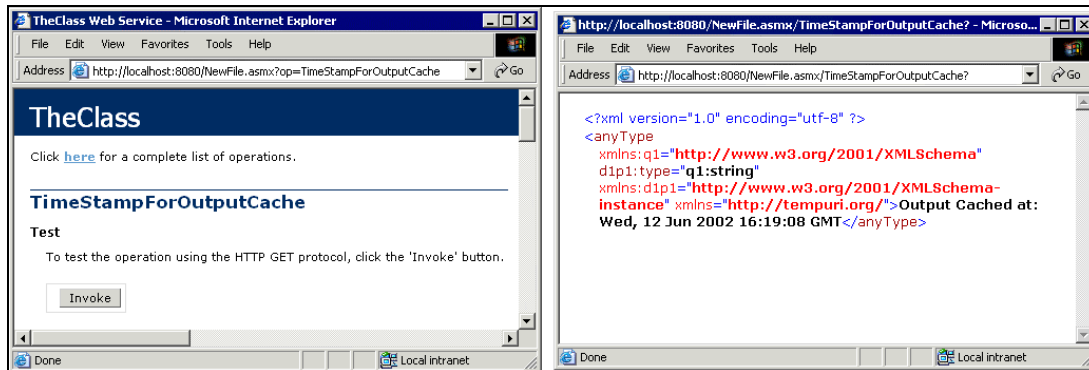
The final section of the New File dialog is the Web Services section. This includes four example pages that implement different features of Web Services. For each one, you must enter the class name and namespace before you can create the file:



- ❑ **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 a @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 screenshot shows the page opened in a browser, and the result:



## Language, Class Names, and Namespaces

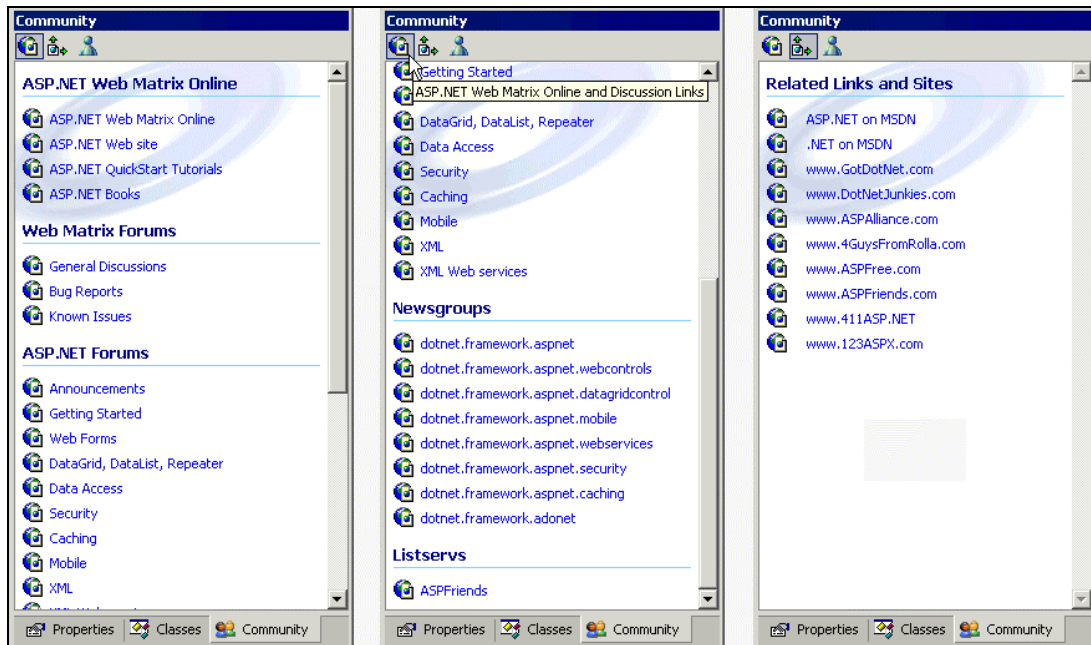
Remember that, for each type of file selected in the New File dialog, any code automatically included in the file is in the language that you specify in that dialog – the choice is between Visual Basic .NET and C#. Depending on the type of file you select, the dialog will also contain controls in which 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 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 combo-box 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.

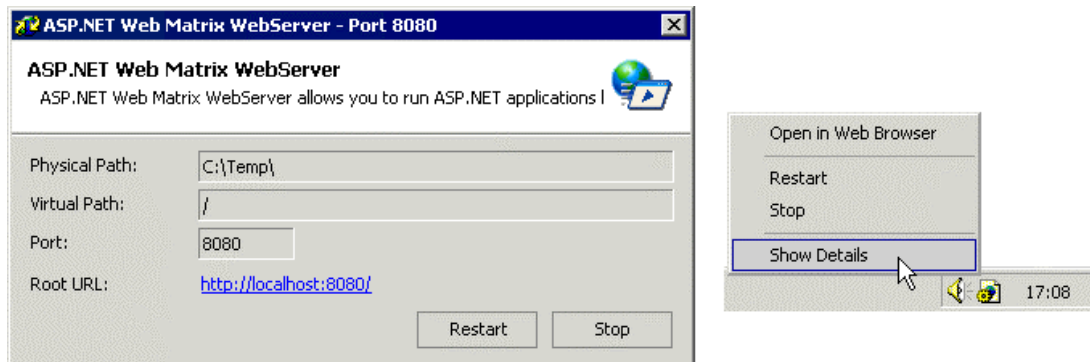
## ***Sending Feedback to Microsoft***

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):



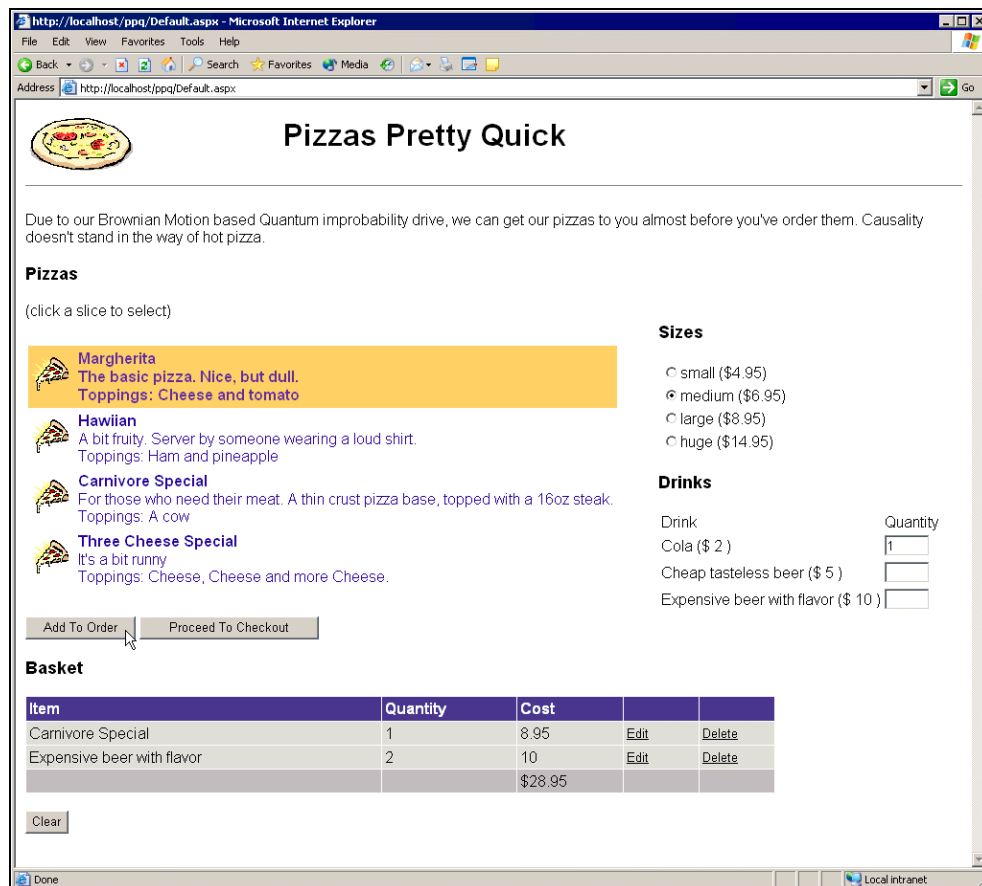


## Part 2 – Putting Web Matrix to Work

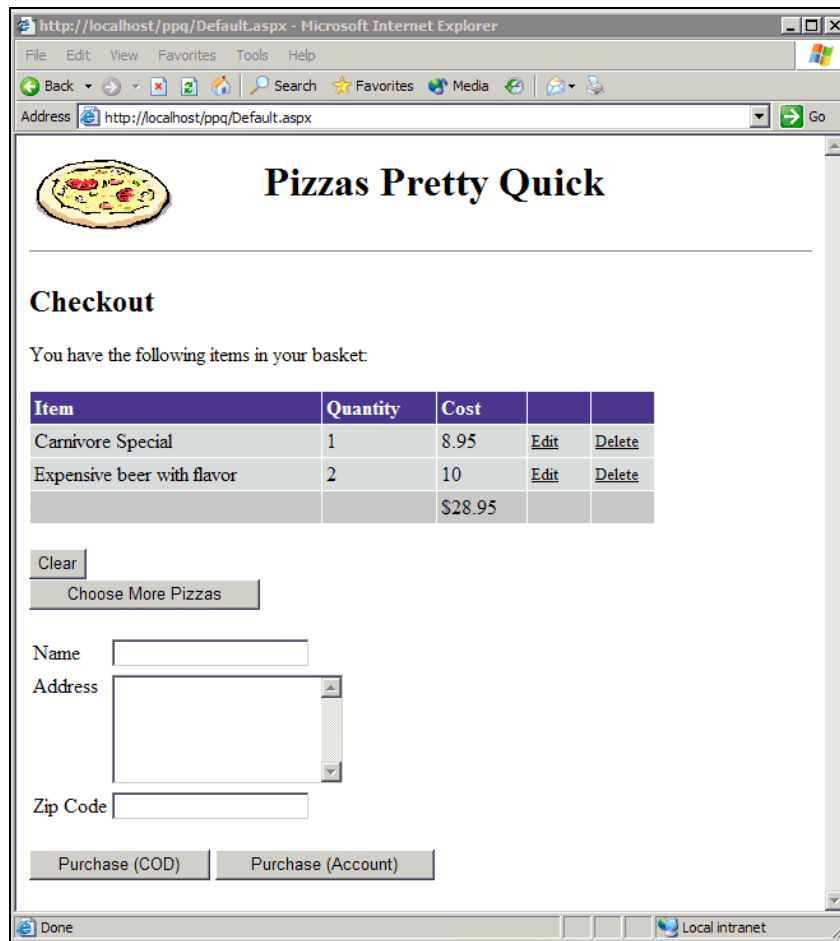
Now that you know what tasks Microsoft ASP.NET Web Matrix is capable of, it's time to put them into practice. Web Matrix is easy to use, so we're not going to show you every aspect of it in action. Instead, we'll build a simple Web site for a pizza delivery company, concentrating on the most commonly 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 redisplay 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 sample is available from <http://www.AIAndDave.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.

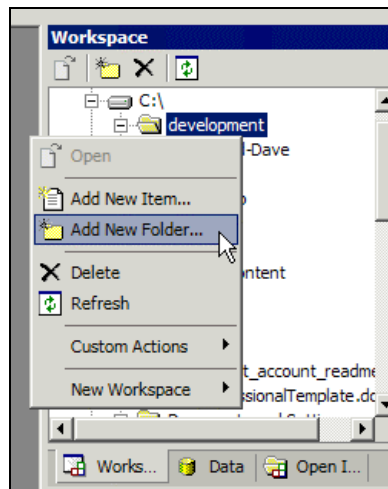
In the following sections we'll tackle:

- ❑ 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 details and how they'd like to pay

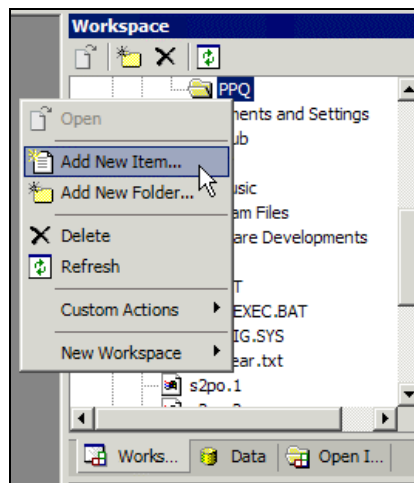
- ❑ Creating secure pages for customers with accounts
- ❑ Creating a variety of different pages, such as those that use a Master and Details grid, or those that require caching
- ❑ How to create Web Services
- ❑ How to use other controls, such as the Internet Explorer controls and custom controls

## 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:

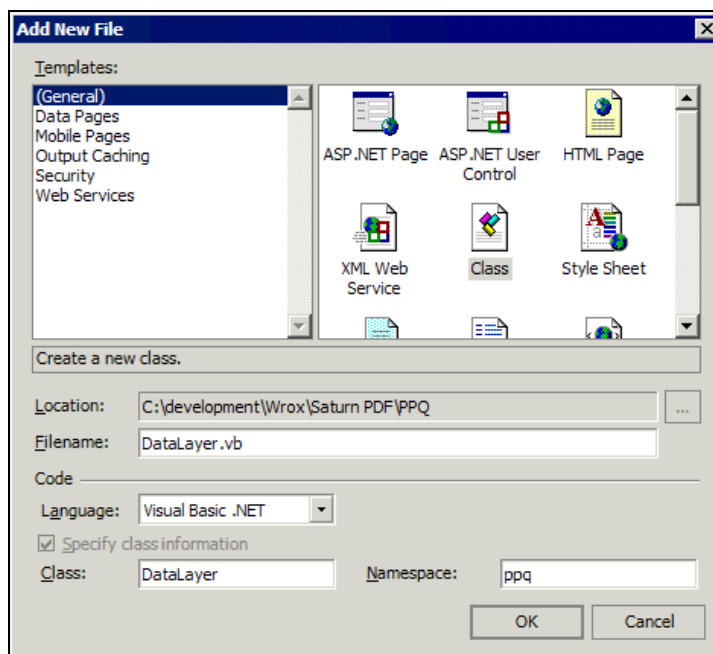
```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <ppq>
3   <pizza name="Margherita" ingredients="Cheese and tomato"
4     description="The basic pizza. Nice, but dull."/>
5   <pizza name="Hawian" ingredients="Ham and pineapple"
6     description="A bit fruity. Server by someone wearing a loud shirt."/>
7   <pizza name="Carnivore Special" ingredients="A cow"
8     description="For those who need their meat. A thin crust pizza base, topped with a 16oz steak."/>
9   <pizza name="Three Cheese Special" ingredients="Cheese, Cheese and more Cheese."
10    description="It's a bit runny"/>
11   <drink name="Cola" price="2"/>
12   <drink name="Cheap tasteless beer" price="5"/>
13   <drink name="Expensive beer with flavor" price="10"/>
14   <size name="small ($4.95)" price="4.95"/>
15   <size name="medium ($6.95)" price="6.95"/>
16   <size name="large ($8.95)" price="8.95"/>
17   <size name="huge ($14.95)" price="14.95"/>
18 </ppq>
19
20

```

In reality it's likely that you'd use a database for all of these details, but this 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

```

This class has two simple methods. The first method simply loads and returns the XML file as a DataSet. The second method accesses a SQL database in order to add details of the order and order lines. All this is fairly standard code that uses a couple of stored procedures and parameters. The important thing to understand about this code is that the shopping basket is held in the current Session – we'll see how that's created later.

### Compiling Classes

One thing that Web Matrix doesn't do is compilation, so we have to perform this manually. We simply created a batch file with the following in it (note that this command is all one line):

```

vbc /debug /nologo /t:library /out:bin/DataLayer.dll /r:System.dll /r:System.Xml.dll
/r:System.Web.dll /r:System.Data.dll Datalayer.vb

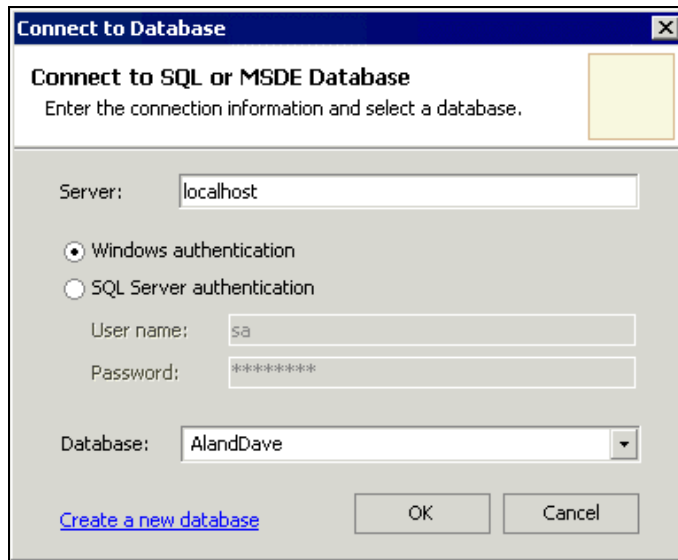
```

Then we can run this batch file from the command line. This sort of thing would actually be quite a good add-in.

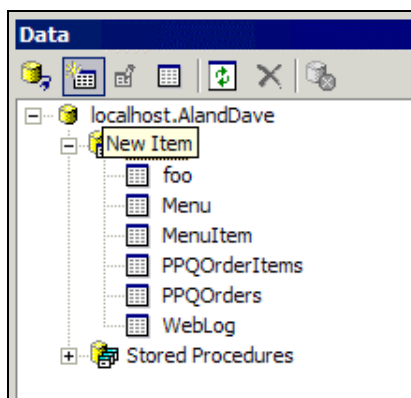


## Managing the Data

We discussed Web Matrix's data management features in *Section 1*, but now we can put these features into practice. First we'll use the Data tab to create a new connection (this assumes you've got a database already created):

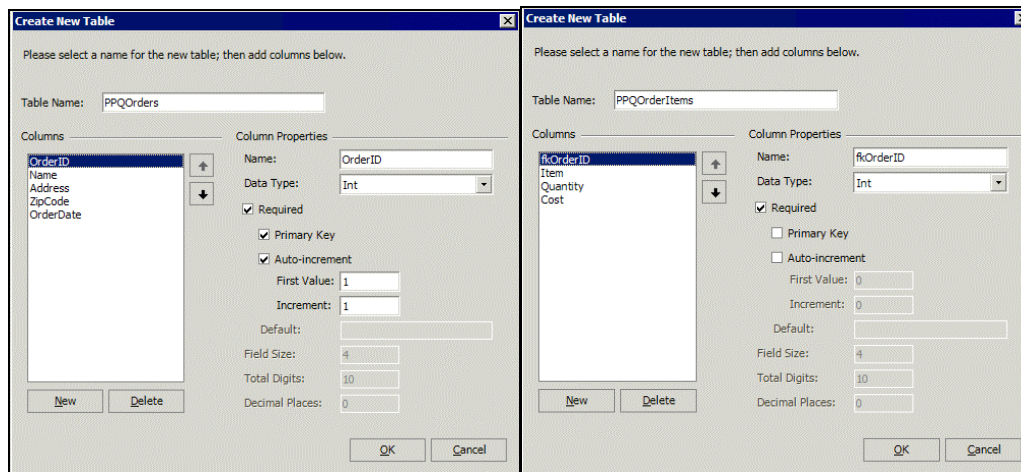


Once we've connected to the database we can create a new table by selecting **Tables** and then clicking the **New Item** button:

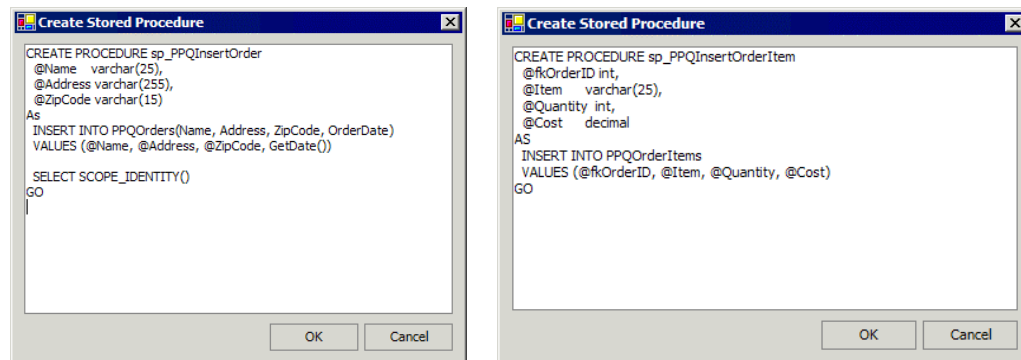


*If you don't want to go through this process, you can use SQL scripts that are provided with the code download to create the tables and stored procedures.*

You'll need to create the following tables:



You'll also need to create the following stored procedures:

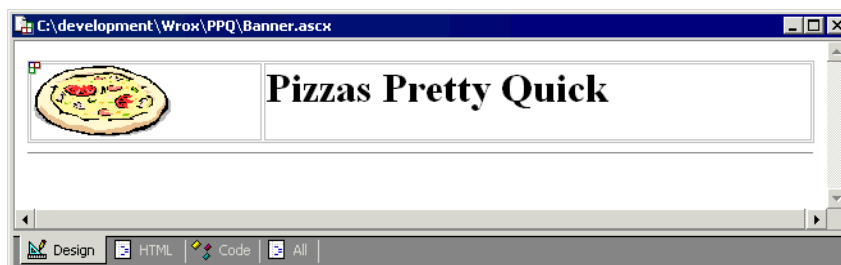


Although the stored procedure window appears to be just a simple text editor but it actually validates the stored procedure as it saves it to SQL Server. However, it doesn't add any required permissions (such as GRANT requests for particular users) so you'll have to add these yourself. The exact form of these permissions will depend on your connection details, and what permissions your user requires.

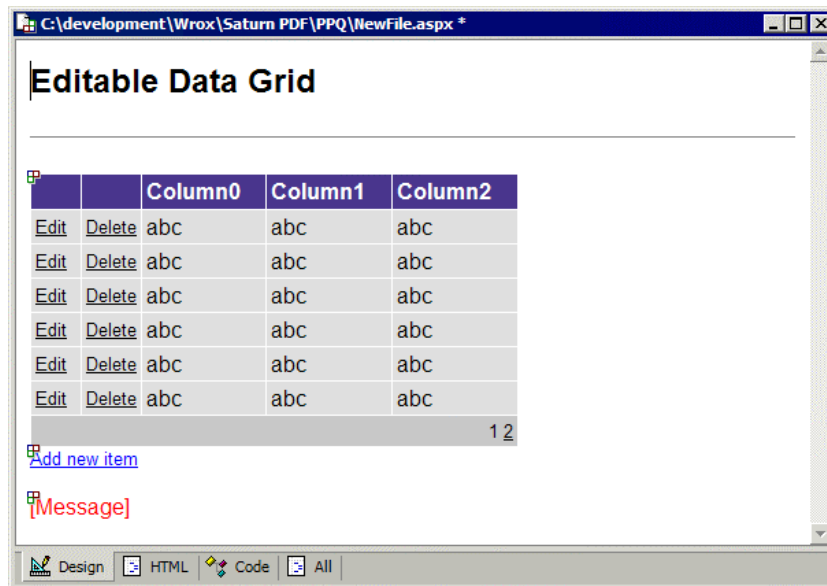
## Creating User Controls

We're going to use two User Controls in our application. The first control is a simple banner at the top of the page, which will contain an image and heading. The second control is for the shopping basket. We'll use a User Control for the shopping basket as it will be used on several pages, and encapsulates quite a lot of functionality.

You can create a User Control by selecting ASP.NET User Control from the New Item dialog. Our first control is very simple, and we can simply drag some HTML controls onto the design surface. The control comprises of an HTML table that contains a single row and two columns, inside of which are an image and some text. There's no need to add any code:



The second User Control is our shopping basket, and since this control requires some database functionality, we won't use the ASP.NET User Control template. Instead we'll use the Editable Data Grid from the Data Pages section of the New Item dialog. Rename the extension of the file to .ascx – we need to do this as we want the shopping basket to be an editable grid. Using the correct template and extension means that Web Matrix generates lots of useful code for us. The following default page is created:



Web Matrix has created an editable grid, along with code that allows us to edit and save data. For example, the following code is provided to allow updates:

```
Sub DataGrid_Update(Sender As Object, E As DataGridCommandEventArgs)

    ' update the database with the new values

    ' get the edit text boxes
    Dim id As String = CType(e.Item.Cells(2).Controls(0), TextBox).Text
    Dim lname As String = CType(e.Item.Cells(3).Controls(0), TextBox).Text
    Dim fname As String = CType(e.Item.Cells(4).Controls(0), TextBox).Text

    ' TODO: update the Command value for your application
    Dim myConnection As New SqlConnection(ConnectionString)
    Dim UpdateCommand As SqlCommand = new SqlCommand()
    UpdateCommand.Connection = myConnection

    If AddingNew = True Then
        UpdateCommand.CommandText = "INSERT INTO authors(au_id, " & _
            "au_lname, au_fname, contract) " & _
            "VALUES (@au_id, @au_lname, @au_fname, 0)"
    Else
        UpdateCommand.CommandText = "UPDATE authors " & _
            "SET au_lname = @au_lname, au_fname = @au_fname " & _
            "WHERE au_id = @au_id"
    End If

    UpdateCommand.Parameters.Add("@au_id", SqlDbType.VarChar, 11).Value = id
    UpdateCommand.Parameters.Add("@au_lname", SqlDbType.VarChar, 40).Value = _
        lname
    UpdateCommand.Parameters.Add("@au_fname", SqlDbType.VarChar, 20).Value = _
        fname

    ' execute the command
    Try
        myConnection.Open()
        UpdateCommand.ExecuteNonQuery()

    Catch ex as Exception
        Message.Text = ex.ToString()

    Finally
        myConnection.Close()

    End Try
```

```

    '' Resort the grid for new records
    If AddingNew = True Then
        DataGrid1.CurrentPageIndex = 0
        AddingNew = false
    End If

    '' rebind the grid
    DataGrid1.EditItemIndex = -1
    BindGrid()

End Sub

```

This code uses the pubs database as its template, and includes a connection string at the top of the page. All we have to do to use this code in our own example is to change a few details to match our database.

You might prefer to take out much of this data access code and replace it with calls to a data layer that performs this functionality for you. If this is something you'll be doing a lot of, then it's worth creating your own templates that suit your style of coding. We'll see how to create our own templates is covered in *Section 3*.

## Modifying the Code

There's not much to do in order to modify the standard template code to work with our own data. We need to modify the connection string and SELECT statement used to fetch the data. Both of these are found at the top of the page:

```

Dim ConnectionString As String = "server=(local);database=pubs;Integrated Security=SSPI"
Dim SelectCommand As String = "SELECT au_id, au_lname, au_fname from Authors"

```

We also need to modify three event handlers:

- ☐ DataGrid\_Update
- ☐ DataGrid\_Delete
- ☐ AddNew\_Click

Let's look at the sort of changes you'll often have to make to these methods. After that, we'll look at the specific changes we need to make for our example application.

### DataGrid\_Update

The following modifications need to be made to the DataGrid\_Update method:

```

'' get the edit text boxes
Dim id As String = CType(e.Item.Cells(2).Controls(0), TextBox).Text
Dim lname As String = CType(e.Item.Cells(3).Controls(0), TextBox).Text
Dim fname As String = CType(e.Item.Cells(4).Controls(0), TextBox).Text

```

This event handler is called when a row is updated, and the DataGridCommandEventArgs object passed into the method points to the row being updated. The data is fetched out of the cells into variables – you'll need to modify these lines to pick your data out of the grid. The first two columns are the edit and delete columns, which is why the data starts at the third column (the Cells collection is zero-based).

Next we need to set the command text:

```

'' TODO: update the Command value for your application
Dim myConnection As New SqlConnection(ConnectionString)
Dim UpdateCommand As SqlCommand = new SqlCommand()
UpdateCommand.Connection = myConnection

If AddingNew = True Then
    UpdateCommand.CommandText = "INSERT INTO authors(au_id, " & _
        "au_lname, au_fname, contract) " & _
        "VALUES (@au_id, @au_lname, @au_fname, 0)"
Else
    UpdateCommand.CommandText = "UPDATE authors " & _

```

```

        "SET au_lname = @au_lname, au_fname = @au_fname " & _
        "WHERE au_id = @au_id"
    End If

```

Again this will need modifying to match your data. Alternatively, you could set the `CommandText` property of the command to the name of a stored procedure, and the `CommandType` to `CommandType.StoredProcedure`.

Next are the parameters for the command. These also need modifying:

```

UpdateCommand.Parameters.Add("@au_id", SqlDbType.VarChar, 11).Value = id
UpdateCommand.Parameters.Add("@au_lname", SqlDbType.VarChar, 40).Value _
    = lname
UpdateCommand.Parameters.Add("@au_fname", SqlDbType.VarChar, 20).Value _
    = fname

```

The rest of the procedure remains the same.

### **DataGrid\_Delete**

For `DataGrid_Delete` all we have to do is change the command that performs the deletion:

```

Dim DeleteCommand As New SqlCommand("DELETE from authors where au_id='" & _
    keyValue & "'", myConnection)

```

Like the update this could also be changed to call a stored procedure, as long as the `CommandType` was set correctly to `CommandType.StoredProcedure`.

### **AddNew\_Click**

For additions the code creates a new array of data that is inserted into the table:

```

'' add a new blank row to the end of the data
Dim rowValues As Object() = {"", "", ""}
ds.Tables(0).Rows.Add(rowValues)

```

In this code the new values (three of them) are strings, so empty strings are used. You'll have to add objects of the correct type that match the type defined in the columns of the table.

## **Modifying Our User Control**

For our control, we simply persist the shopping cart details in a `DataTable` in the `Session`, so all of the database access code can be removed. For example, the grid update routine is now:

```

Sub DataGrid_Update(Sender As Object, E As DataGridCommandEventArgs)

    '' get the edit text boxes
    Dim id As Integer = CInt(ShoppingBasket.DataKeys(e.Item.ItemIndex))
    Dim qty As String = CType(e.Item.Cells(1).Controls(0), TextBox).Text

    ChangeQuantity(id, qty)

    '' rebind the grid
    ShoppingBasket.EditItemIndex = -1
    BindGrid()

End Sub

```

The other data functions are also modified to use methods that manipulate the shopping basket items. These methods have been made public so that they can be called from the page that hosts the user control. We won't go into the detail of all of these methods, since they are fairly easy to understand, and aren't Web Matrix-specific.

One thing that is worth mentioning is a property for the basket we create:

```

Public WriteOnly Property ViewMode As Boolean
    Set
        If Value = True Then
            btnClear.Visible = False
            Dim cols As Integer = ShoppingBasket.Columns.Count
            ShoppingBasket.Columns(cols-1).Visible = False
            ShoppingBasket.Columns(cols-2).Visible = False
        End If
    End Set
End Property

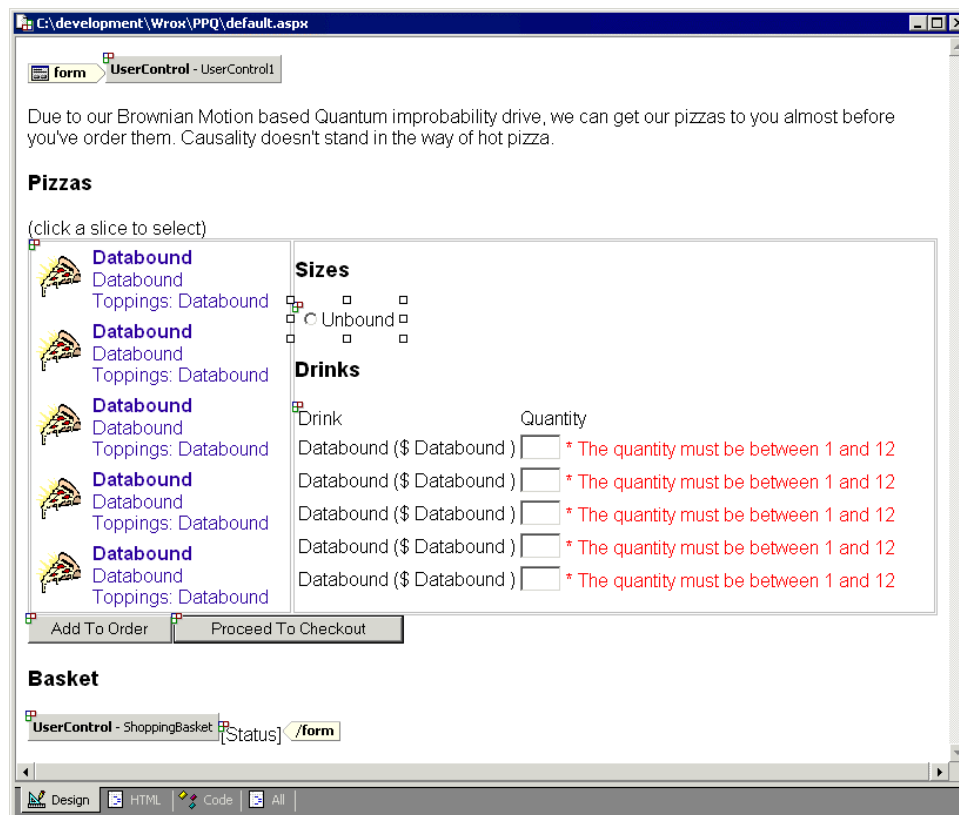
```

The setter for this property turns the editable grid into a read-only grid. It does this by making the edit column, the delete column, and the clear button invisible. Our User Control can now be used on multiple pages in both editable and non-editable modes.

We now have a fully functional shopping basket that can be dropped onto other pages.

## The Main Page

Although our main default page will display data, the data templates provided by Web Matrix all produce grids, and we want a combination of different data controls. So, we'll start with a blank ASP.NET page instead. Once we've added a few controls the page looks like this:



There's a combination of quite a few controls here. At the top of the page we have the User Control we created earlier that represents the banner. Web Matrix doesn't provide design time support for these sorts of controls, and there's no way to drag them onto the design surface. To add them to a page we have to use the All view, so that we can add both the @Register directive and the control:

```
<%@ Register TagPrefix="ppq" TagName="Banner" Src="Banner.ascx" %>
```

```
<ppq:banner id="UserControl1" runat="server"></ppq:banner>
```

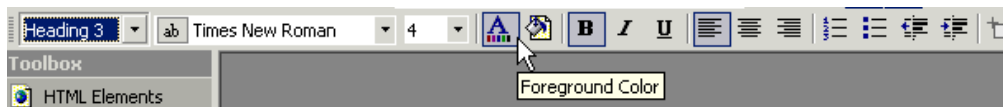
We need to repeat the process for the shopping basket:

```
<%@ Register TagPrefix="ppq" TagName="ShoppingBasket"
    Src="ShoppingBasket.ascx" %>
```

```
<ppq:ShoppingBasket id="ShoppingBasket" runat="server">
</ppq:ShoppingBasket>
```

If you switch back into Design view then you'll see that the user controls are displayed as gray panels.

You can add the text and labels to the page simply by dragging and dropping them from the Toolbox. Then you can set their properties as required. Web Matrix contains a formatting toolbar, so you don't even have to remember what any of the formatting attributes are:



For the actual data, we want to show three things:

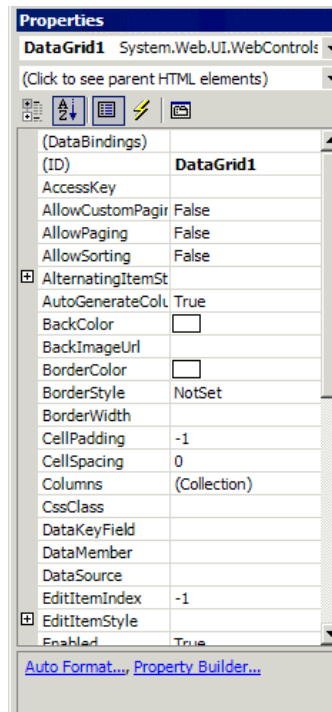
- ☐ The selection of pizzas
- ☐ What sizes they come in
- ☐ Available drinks

Each of these requires a different data control: the pizzas will be displayed using a DataGrid, the sizes using a RadioButtonList, and the drinks using a Repeater with custom content. In order to format them nicely the controls are contained within an HTML table. Dragging a table onto the design surface provides, by default, a table that has three rows and three columns, so we need to edit the code in HTML view in order to delete two of the rows.

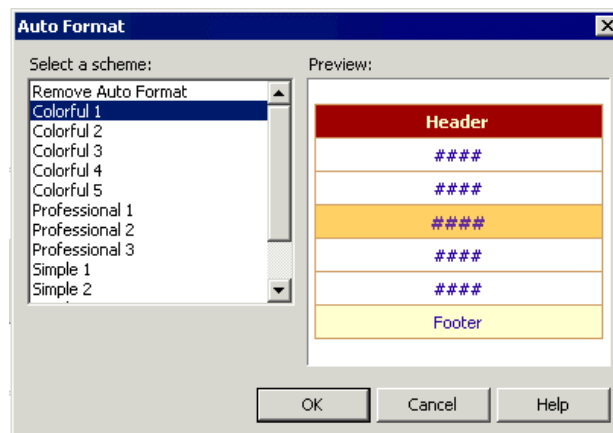
So that we can display the selection of pizzas, we drag a DataGrid into the first cell of the table:

Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

The Auto Format and the Property Builder links, found at the bottom of the properties panel, can be used to format and customize the DataGrid:

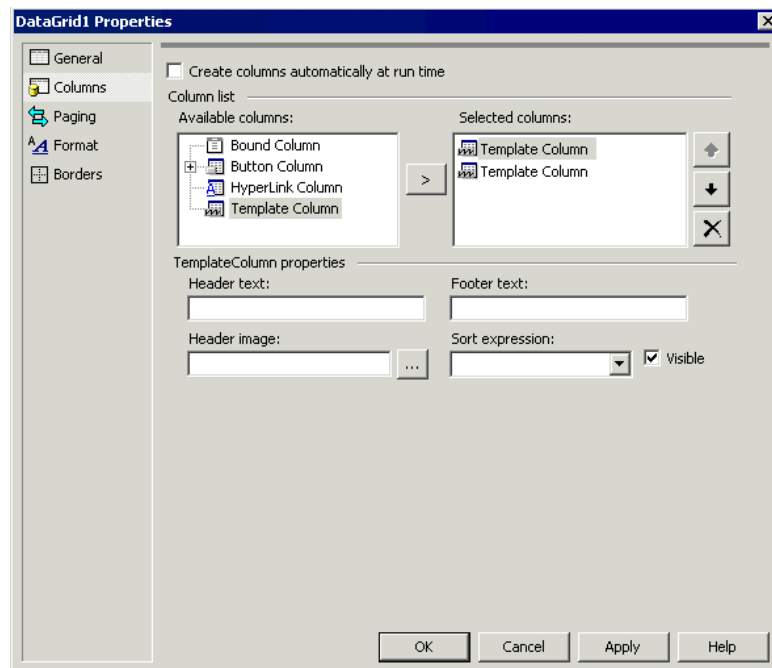


These links provide access to the same builders that Visual Studio .NET uses. The Auto Format builder allows us to pick a visual style; the designer will set the style properties of the grid for us:



The Property Builder allows us, among other things, to set the column details:





Here we've stated that we don't want the columns automatically generated, and we've added two `TemplateColumns`. Closing the dialog and switching to HTML view allows us to see what this has done:

```
<asp:DataGrid id="DataGrid1" runat="server" BorderStyle="None"
    BorderWidth="1px" BorderColor="#CC9966" BackColor="White"
    CellPadding="4" AutoGenerateColumns="False">
  <FooterStyle forecolor="#330099" backcolor="#FFFFCC"></FooterStyle>
  <HeaderStyle font-bold="True" forecolor="#FFFFCC" backcolor="#990000">
</HeaderStyle>
  <PagerStyle horizontalalign="Center" forecolor="#330099"
    backcolor="#FFFFCC"></PagerStyle>
  <SelectedItemStyle font-bold="True" forecolor="#663399"
    backcolor="#FFCC66"></SelectedItemStyle>
  <ItemStyle forecolor="#330099" backcolor="White"></ItemStyle>
  <Columns>
    <asp:TemplateColumn></asp:TemplateColumn>
    <asp:TemplateColumn></asp:TemplateColumn>
  </Columns>
</asp:DataGrid>
```

Now we can add our details into the template columns. There are two ways of doing this. The first is by simply switching to HTML view and typing in the details:

```
<Columns>
  <asp:TemplateColumn>
    <ItemTemplate>
      <asp:ImageButton CommandName="Select"
        ImageURL="images/slice.gif" height="40" width="40"
        CausesValidation="false" runat="server"/>
    </ItemTemplate>
  </asp:TemplateColumn>
  <asp:TemplateColumn>
    <ItemTemplate>
      <b><asp:Label id="pizza"
        Text=' '<# DataBinder.Eval(Container.DataItem, "name") %>'
        runat="server" />
      </b>
      <br />
      <# DataBinder.Eval(Container.DataItem, "description") %><br />
      Toppings:
    </ItemTemplate>
  </asp:TemplateColumn>
</Columns>
```

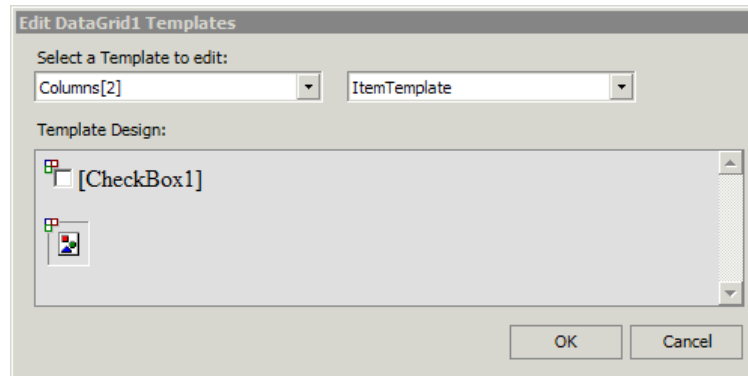
```

        <%# DataBinder.Eval(Container.DataItem, "ingredients") %>
    </ItemTemplate>
</asp:TemplateColumn>
</Columns>

```

The first column is an image button, which acts as our selected method. The second column shows the details of the pizza.

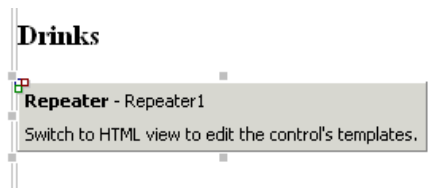
Alternatively, you can use the Template Editor, which is accessed from the **Edit Template...** menu item from the **Edit** menu when you have the **DataGrid** selected in **Design** view:



This dialog gives you the option to select the column and template, and then you can drag controls from the Toolbox into the Template Design surface.

In order to display the pizza sizes we'll add a **RadioButtonList** from the Toolbox into the second cell of the table. We'll add the data binding in code, so there's nothing more to do for this control for the moment.

To display the selection of drinks we'll use a **Repeater** control:



The **Repeater** is a look-less control, which means that it has no UI editing features. Instead, the look is created using templates in **HTML** view:

```

<asp:Repeater id="Drinks" runat="server">
    <HeaderTemplate>
        <table>
            <tr>
                <td>Drink</td>
                <td>Quantity</td>
            </tr>
        </HeaderTemplate>
    <ItemTemplate>
        <tr>
            <td>
                <asp:Label id="Drink"
                    Text=' '<%# DataBinder.Eval(Container.DataItem, "name") %>'
                    runat="server" />

                ($
                <asp:Label id="DrinkCost"
                    Text=' '<%# DataBinder.Eval(Container.DataItem, "price") %>'
                    runat="server" />
                )
            </td>
            <td>
                <asp:TextBox id="DrinkQuantity" columns="2" runat="server" />
            </td>
        </tr>
    </ItemTemplate>
</asp:Repeater>

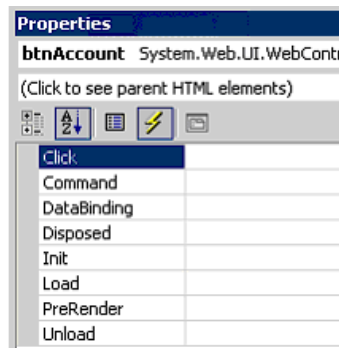
```

```

        <asp:RangeValidator ControlToValidate="DrinkQuantity"
            MinimumValue="1" MaximumValue="12" Type="Integer"
            ErrorMessage="** The quantity must be between 1 and 12"
            Display="Dynamic" runat="server" />
    </td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</td>

```

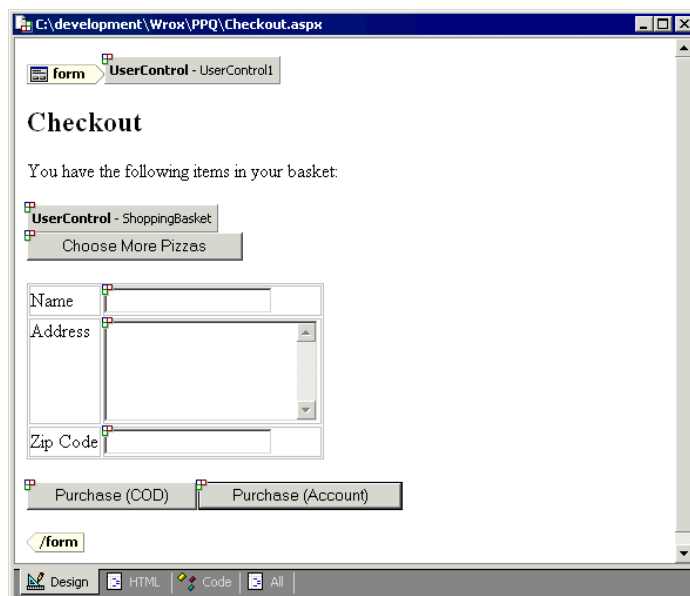
Finally we need to add two buttons to the form. The first adds the items a customer has selected to the shopping basket, and the second takes the customer to the checkout page. Creating the event handlers for these is simple – we can either double-click the button, which takes us to the code editor, or we can use the events listed on the Properties pane:



From here we can select existing event procedures, or double-click within an event name in order to create the event handler and be placed in the event window.

## The Checkout Page

The checkout page is fairly simple and, as we did with the default main page, we'll start with a blank ASP.NET page that we can add controls to:



Here we've used the same user controls as we did in our main page, in addition to text entry fields for the delivery details. The customer has the option of placing the order and paying cash when it is delivered, or logging into a secure area of the site where they can add the balance to their account.

Of particular interest here is that this page exposes three custom properties:

```
Public ReadOnly Property DeliveryName As String
    Get
        Return txtDeliveryName.Text
    End Get
End Property

Public ReadOnly Property DeliveryAddress As String
    Get
        Return txtDeliveryAddress.Text
    End Get
End Property

Public ReadOnly Property DeliveryZipCode As String
    Get
        Return txtDeliveryZipCode.Text
    End Get
End Property
```

These properties ensure that the delivery information is available to the delivery page. We have to do it this way, since the viewstate security features prevent 'us from posting to another page and reading the details from the form. We could transfer to another page and pass these details as part of the query string, but I think that's an ugly solution. It's far better to use the supplied framework and expose the content as properties that can be accessed in the next page. For this to work, we also need to add a class name to the current page:

```
<%@ Page Language="VB" " ClassName=" "Checkout" " %>
```

If you know in advance that your page is going to need a class name, then you can set this in the **New File** dialog.

## ***The Delivery Page***

The delivery page serves just to confirm the delivery details for an order. We show the shopping basket on this page, and this time set the **ViewMode** property we created earlier on the user control to **True**, so that the basket is non-editable:

```
Dim chk As Checkout

Sub Page_Load()

    If Not IsPostBack Then
        chk = CType(Context.Handler, Checkout)
        ShoppingBasket.ViewMode = True
    End If

End Sub
```

We also reference the preceding page – this is why we gave it a class name, so that we can get the delivery details by just referencing the properties:

```
Delivery to:
<br />
<% = chk.DeliveryName %>
<br />
<% = chk.DeliveryAddress %>
<br />
<% = chk.DeliveryZipCode %>
<br />
```

Finally we empty out the shopping basket:

```
Sub Page_Unload()

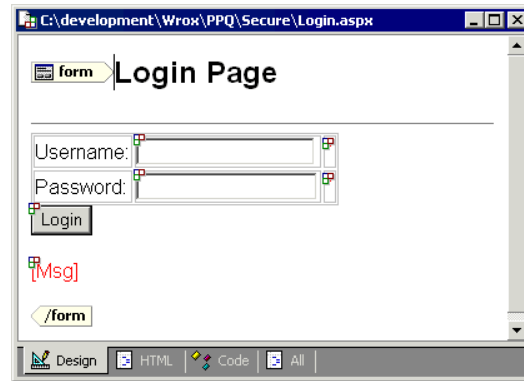
    Session("Basket") = Nothing

End Sub
```

## Adding a Login Page

Security under ASP.NET is a fairly large topic, and it's certainly not my intention to cover it here. However, in order to add security pages to the application, we must understand that security works at the application level, not on a page-by-page basis. This means that we have to create a secure area in a separate directory (I called it *secure*), and mark this directory as an application using the IIS administration tool.

Into this new directory we add a login page, picked from the Security templates, which are available from the New Item dialog:



The code that Web Matrix generates for this page is extremely simple:

```
Sub LoginBtn_Click(Sender As Object, E As EventArgs)

    If Page.IsValid Then
        If (UserName.Text = "jdoe@somewhere.com") And _
            (UserPass.Text = "password") Then
            FormsAuthentication.RedirectFromLoginPage(UserName.Text, true)
        Else
            Msg.Text = "Invalid Credentials: Please try again"
        End If
    End If

End Sub
```

This event handler uses a fixed name and password, so we'll need to customize it:

```
Sub LoginBtn_Click(Sender As Object, E As EventArgs)

    If Page.IsValid Then
        If FormsAuthentication.Authenticate(UserName.Text, UserPass.Text) Then
            FormsAuthentication.RedirectFromLoginPage(UserName.Text, true)
        Else
            Msg.Text = "Invalid Credentials: Please try again"
        End If
    End If

End Sub
```

Here we're just taking the details from the screen and passing them into the `Authenticate` method. How the authentication takes place depends on how the security is configured. This configuration is done using the application configuration file (named `web.config`), so we add a `web.config` file to our secure area, using the one provided in the Security templates.

The default `web.config` template contains all of the possible sections commented out, so you don't have to head to the documentation to look up the details. There are several ways in which we could set the configuration, but we'll use the simplest; we'll store the user names and passwords in the `credentials` section:

```
<authentication mode="Forms">
  <forms name="ppq" path="/" loginUrl="login.aspx"
    protection="All" timeout="30">
  <credentials passwordFormat="Clear">
    <user name="billjones" password="test" />
  </credentials>
  </forms>
</authentication>
```

```

        <user name="marthasmith" password="test" />
        <user name="joesoap" password="test" />
    </credentials>
</forms>
</authentication>

<authorization>
    <allow users="billjones,marthasmith,joesoap" />
    <deny users="?" />
</authorization>

```

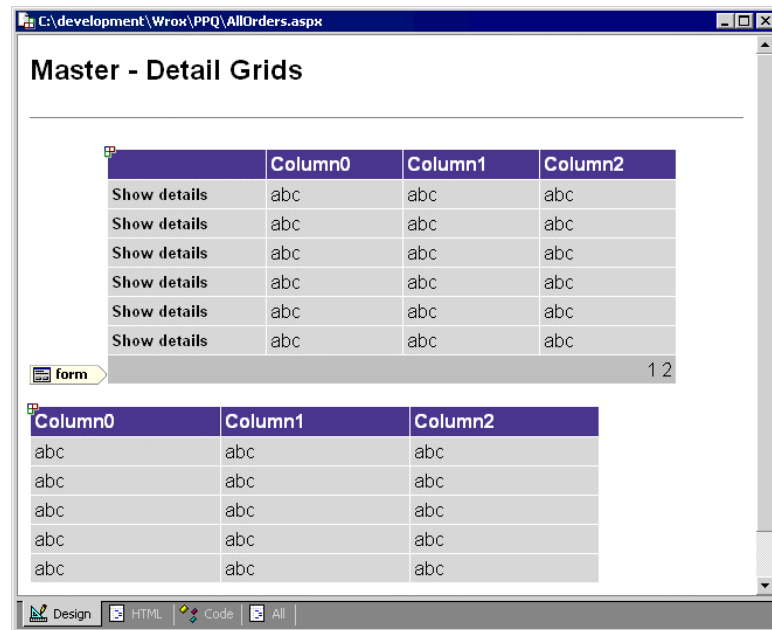
**Obviously in a real site you wouldn't want to use something this insecure (you especially wouldn't want to store the passwords in plain text), but this arrangement suffices for our example.**

We've specified Forms authentication, so if a user tries to access any page in this separate directory they will be redirected to the login page if they aren't logged in.

When we run the login code, the `FormsAuthentication` class will use the credentials in the configuration file to determine if a user is authorized to view the page. If they are authorized, then we can redirect them back to the page they asked for. This way we don't actually have to check to see whether the user is logged in – ASP.NET handles that for us.

## Master – Details Grid

Web Matrix has many other template pages that help reduce the amount of code we have to write, although they'll often require a small amount of customization before they fit our needs exactly. For example, consider a page that shows all of the orders and order details; we could use the Master – Detail Grids template for such a page:



We only have to change a few lines of code in this page to get it to do what we want. In the `BindMasterGrid` procedure, there are two lines that we need to modify (I've split them over multiple lines to make it easier to read):

```

Dim ConnectionString As String = "server=(local);database=pubs;" & _
    "Integrated Security=SSPI"
Dim CommandText As String = "select au_lname as [Last Name], " & _
    "au_fname as [First Name], Address, City, " & _
    "State from Authors order by [Last Name]"

```

We change these lines to:

```
Dim ConnectionString As String = "server=(local);database=AlandDave;" & _
    " Trusted_Connection=true "
Dim CommandText As String = "select * from PPQOrders order by OrderDate"
```

We also need to make a small modification to the DataGrid definition; we need to change the DataKeyField property from Last Name to OrderID:

```
<asp:datagrid id="MasterGrid" DataKeyField="OrderID" . . .
```

This change can be done either in HTML view, or via the Properties box in design view.

We also need to modify a few lines in the BindDetailGrid procedure. We need to change this:

```
' TODO: update the ConnectionString value for your application
Dim ConnectionString As String = "server=(local);database=pubs;" & _
    "Integrated Security=SSPI"


' TODO: update the CommandText value for your application
Dim filterValue As String = _
    CStr(MasterGrid.DataKeys(MasterGrid.SelectedIndex)).Replace("'", "")
Dim CommandText As String = "select title as Title, price as Price, " & _
    "ytd_sales as [YTD Sales] from titleview " & _
    "where au_lname = '" & filterValue & "'" "
```

to this:

```
Dim ConnectionString As String = "server=(local);database=AlandDave;" & _
    " Trusted_Connection=true "

Dim filterValue As String = CStr(MasterGrid.DataKeys(MasterGrid.SelectedIndex))
Dim CommandText As String = "select Item, Quantity, Cost " & _
    "from PPQOrderItems where fkOrderID=" & filterValue
```

That's all we need to change – we now have a fully functioning Master – Detail Grid:

Address  http://localhost/ppq/allorders.aspx

### Master - Detail Grids

	OrderID	Name	Address	ZipCode	OrderDate
Show details	4	Dave	here	there	4/10/2002 13:03:32
Show details	5	A.N. Other	elsewhere		4/11/2002 16:18:37
Show details	6	Dave	Hungry House		4/11/2002 16:18:59
Show details	7	him	there		4/11/2002 16:19:23
Show details	8	me again			4/11/2002 16:20:09

1

Item	Quantity	Cost
Carnivore Special	1	15
Expensive beer with flavo	2	10
Three Cheese Special	1	5
Cola	1	2
Hawaiian	1	5
Cheap tasteless beer	3	5
Margherita	1	9

Although this page is fairly basic, it can be further customized. For example, we could remove the OrderID column and show line and order totals with little additional effort.

## Cached Pages

The Web Matrix templates for Output Caching all follow a familiar theme. The first is the Vary By None template, which produces the following code:

```
<%@ Page Language="VB" %>
<%@ outputcache duration="10" varybyparam="none" %>

<script runat="server">
    Sub Page_Load(Sender As Object, E As EventArgs)

        TimestampCreated.Text = DateTime.Now.ToString("r")
        TimestampExpires.Text = DateTime.Now.AddSeconds(10).ToString("r")

    End Sub
</script>

<html>
<body style="font-family:arial">
    <h2>
        Output caching for 10 seconds...
    </h2>
    <hr size="1" />
    <p>
        Output Cache created:
        <asp:Label id="TimestampCreated" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
        <br />
        Output Cache expires:
        <asp:Label id="TimestampExpires" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
    </p>
</body>
</html>
```

This template contains two labels that show the current time and the time the cache expires. The cache time is set by the outputcache directive at the top of the page:

```
<%@ outputcache duration="10" varybyparam="none" %>
```

The value of the duration attribute specifies the time in seconds for which this page is to be cached, and the value of the varybynone attribute indicates that there are no parameters to use to define when the page should be cached.

The Vary By Browser template is similar in content:

```
<%@ Page Language="VB" %>
<%@ outputcache duration="10" varybyparam="none" varybycustom="browser" %>

<script runat="server">
    Sub Page_Load(Sender As Object, E As EventArgs)

        BrowserDetails.Text = Request.Browser.Browser & " " & _
            Request.Browser.MajorVersion.ToString()
        TimestampCreated.Text = DateTime.Now.ToString("r")
        TimestampExpires.Text = DateTime.Now.AddSeconds(10).ToString("r")

    End Sub
</script>

<html>
<body style="font-family:arial">
    <h2>
        Vary Cache by Browser
    </h2>
    <hr size="1" />
    <p>
```



```

Varying output by browser:
<asp:Label id="BrowserDetails" ForeColor="red" Font-Bold="true"
    runat="server"></asp:Label>
<br />
Output Cache created:
<asp:Label id="TimestampCreated" ForeColor="red" Font-Bold="true"
    runat="server"></asp:Label>
<br />
Output Cache expires:
<asp:Label id="TimestampExpires" ForeColor="red" Font-Bold="true"
    runat="server"></asp:Label>
</p>
</body>
</html>

```

The same labels as in the Output Caching template are present, together with another that shows the browser details. The `outputcache` directive also has the `varybycustom` attribute set to `browser`, so that different browsers will receive different cached versions of the page.

The Vary Cache By Headers template allows the output to be cached depending on specific HTTP headers. The Web Matrix-generated code follows:

```

<%@ Page Language="VB" %>
<%@ outputcache duration="10" varybyparam="none"
    varybyheader="Accept-Language" %>

<script runat="server">
    Sub Page_Load(Sender As Object, E As EventArgs)

        HeaderDetails.Text = Request.Headers("Accept-Language")
        TimestampCreated.Text = DateTime.Now.ToString("r")
        TimestampExpires.Text = DateTime.Now.AddSeconds(10).ToString("r")

    End Sub
</script>

<html>
<body style="font-family:arial">
    <h2>
        Vary Cache By Headers
    </h2>
    <hr size="1" />
    <p>
        Varying output on header:
        <asp:Label id="HeaderDetails" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
        <br />
        Output Cache created:
        <asp:Label id="TimestampCreated" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
        <br />
        Output Cache expires:
        <asp:Label id="TimestampExpires" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
    </p>
</body>
</html>

```

Instead of showing browser version details, this page shows the `Accept-Language` header, which is the language supported by the browser. The `outputcache` directive has the `varybyparam` attribute set to `none`, but also the `varybyheader` attribute set to `Accept-Language`. This means that there will be separately cached pages for each different language used by browsers.

The final cache template is Vary Cache By Parameters, which caches according to values from the page. The Web Matrix-generated code follows:

```

<%@ Page Language="VB" %>
<%@ outputcache duration="120" varybyparam="Category" %>

<script runat="server">
    Sub Page_Load(Sender As Object, E As EventArgs)

        TimestampCreated.Text = DateTime.Now.ToString("r")
        TimestampExpires.Text = DateTime.Now.AddSeconds(10).ToString("r")

    End Sub

    Sub Button1_Click(Sender As Object, E As EventArgs)

        CategoryItem.Text = "You selected: " & Category.SelectedItem.Text

    End Sub
</script>

<html>
<body style="font-family:arial">
    <form runat="server">
        <h2>
            Vary Cache By Parameters
        </h2>
        <asp:Label id="CategoryItem" runat="server"></asp:Label>
        <hr size="1" />
        Category:
        <asp:DropDownList id="Category" runat="server">
            <asp:ListItem value="default">-- Select Category --</asp:ListItem>
            <asp:ListItem>psychology</asp:ListItem>
            <asp:ListItem>business</asp:ListItem>
            <asp:ListItem value="Popular Computer">popular_comp</asp:ListItem>
        </asp:DropDownList>
        <asp:Button ID="Button1" Text="Lookup" OnClick="Button1_Click"
            runat="server"></asp:Button>
    <p>
        Output Cache created:
        <asp:Label id="TimestampCreated" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
        <br />
        Output Cache expires:
        <asp:Label id="TimestampExpires" ForeColor="red" Font-Bold="true"
            runat="server"></asp:Label>
    </p>
    </form>
</body>
</html>

```

In this page the varybyparam property is set to Category, which is the ID of a DropDownList. Whenever the page is posted back to the server the value stored in Category is used to determine whether the page should be served from the cache or not. You can use any control ID, or multiple control IDs.

## Building ASP.NET Web Services

Web Services provide a way to expose programming functionality across the Web, and their use in Web Matrix is just as easy as other types of ASP.NET pages. There are no advanced features in Web Matrix for Web Service usage, but there are templates provided to make their creation easy.

### Simple and Cached Web Services

The two simplest templates are Simple and Output Caching. The Simple template produces the following class:

```

<%@ WebService language="VB" class="menu" %>

Imports System
Imports System.Web.Services

```

```
Imports System.Xml.Serialization

Public Class menu

    <WebMethod> Public Function Add(a As Integer, b As Integer) As Integer
        Return a + b
    End Function

End Class
```

This provides a default method decorated with the `WebMethod` attribute. The page generated by Web Matrix using the Output Caching template is similar:

```
<%@ WebService language="VB" class="menuCache" %>

Imports System
Imports System.Web.Services
Imports System.Xml.Serialization

Public Class menu

    <WebMethod(CacheDuration:=30)> Public Function TimestampForOutputCache()
        Return "Output Cached at: " & DateTime.Now.ToString("r")
    End Function

End Class
```

Here the `WebMethod` attribute also specifies the duration that the page should be cached for.

## Custom SOAP Headers in a Web Service

The SOAP Headers template contains the following code (it uses a class name and namespace specified when you create the file):

```
<%@ WebService language="VB" class="menuSOAP" %>

Imports System
Imports System.Web.Services
Imports System.Xml.Serialization
Imports System.Web.Services.Protocols

Public Class SimpleHeader
    Inherits SoapHeader

    Public Value As String
End Class

Public Class menuSOAP
    Public soapHeader As SimpleHeader

    <WebMethod, SoapHeader("soapHeader")> _
    Public Function GetValueOfSoapHeader()
        If (soapHeader Is Nothing)
            Return "SOAP Header is empty"
        Else
            Return soapHeader.Value
        End If
    End Function

End Class
```

The code uses a class (inherited from `SoapHeader`) that provides the custom details for the SOAP header. A public method, `GetValueOfSoapHeader`, accesses the custom class and returns the items that are required to be part of the header.

## Custom Class-based Web Service

Web Services can also be based on custom classes. The code contained in the Web Matrix Custom Class template uses `OrderDetails` and `OrderItem` classes to detail orders:

```

<%@ WebService language="VB" class="menuCustom" %>

Imports System
Imports System.Web.Services
Imports System.Xml.Serialization
Imports System.Web.Services.Protocols

Public Class menuCustom

    <WebMethod> Public Function GetOrderDetails()
        Dim orderDetails As New OrderDetails()

        ' Set values for OrderDetails class
        orderDetails.OrderNumber = 35
        orderDetails.CompanyName = "ACME Paint"
        orderDetails.CustomerFirstName = "John"
        orderDetails.CustomerLastName = "Smith"
        orderDetails.CustomerEmail = "John.Smith@IBuySpy.com"

        ' Return an array of 2 OrderItems
        orderDetails.OrderItems As New OrderItem[1];

        orderDetails.OrderItems[0].ItemName = "Sunset Yellow"
        orderDetails.OrderItems[0].ItemId = 12
        orderDetails.OrderItems[0].ItemName = "at the beach"

        orderDetails.OrderItems[1].ItemName = "Seattle Sky Blue"
        orderDetails.OrderItems[2].ItemId = 93
        orderDetails.OrderItems[3].ItemName = "A rare shade of blue"

        Return orderDetails
    End Function

End Class

Public Class OrderDetails

    ' Serialize as an attribute named OrderId
    <XmlAttribute("OrderId")> Public OrderNumber As Integer

    ' Serialize as an attribute
    <XmlAttribute()> Public CompanyName As String

    Public CustomerFirstName As String

    Public CustomerLastName As String

    ' Serialize as an element, but change the name
    <XmlElement("email")> Public CustomerEmail As String

    ' Rename the array OrderItemDetail
    <XmlArray("OrderItemDetail")> Public OrderItem[] OrderItems
End Class

Public Class OrderItem

    ' Serialize all member variables as attributes

    <XmlAttribute()> Public ItemName As String

    <XmlAttribute()> Public ItemId As Integer

    <XmlAttribute()> Public ItemDescription As String
End Class

```

Both classes (OrderDetails and OrderItem) have public variables that are decorated with attributes that specify how the property should be serialized. The Web Method of the custom class, GetOrderDetails, returns an instance of the OrderDetails class, which is serialized according to the specified attributes.

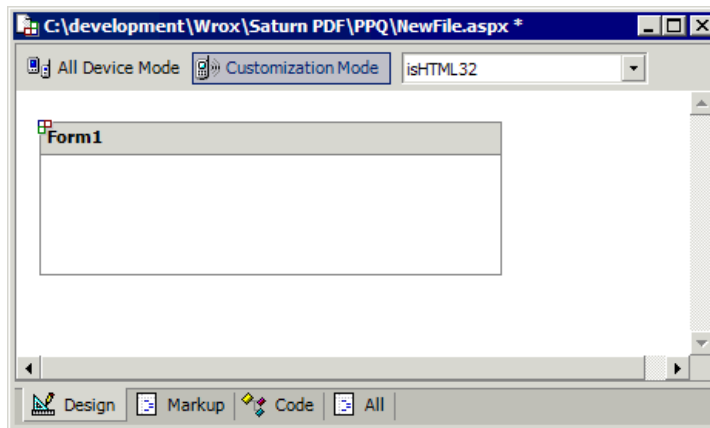
## Using Other Components

In addition to the standard HTML and ASP.NET Server controls, there are plenty of other controls that can be used with Web Matrix. The market for third-party controls is growing quickly, and these can be added to Web Matrix by customizing the Toolbox. You can even create your own custom controls and add them to the toolbox. There are also two additional sets of controls supplied by Microsoft: the Mobile Internet Toolkit Controls for creating applications for mobile devices and the Internet Explorer Web Controls for creating rich content for IE.

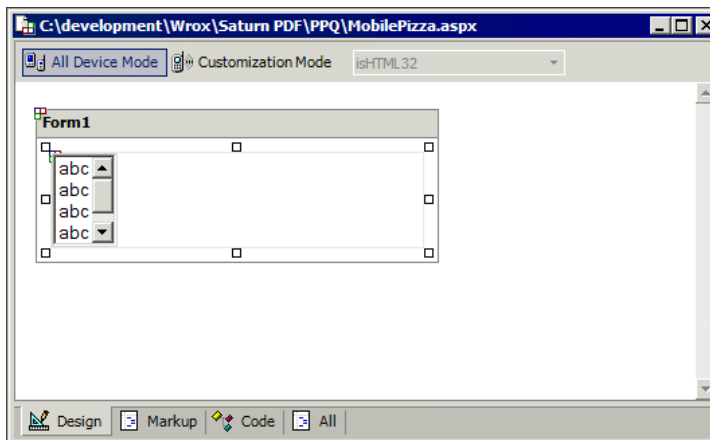
Web Matrix uses the same control designer model as Visual Studio .NET, so controls will work in both tools.

### Mobile Internet Toolkit Controls

Support for mobile devices is just as good as for traditional browser pages, but you will need to install the Mobile Internet Toolkit first. You have the choice of a Simple Mobile Page or a Simple Mobile User Control, both of which look like this:



By default this page will support all mobile devices, but you can select the Customization Mode, which allows you to target specific implementations of devices. Although the designer differs in look, its use is exactly the same as for other pages. You drag and drop the mobile controls onto the design surface, and then add your own code. For example, we can add a selection list for pizzas:



Here we've added a `SelectionList` control to the form. Then, we've set its `SelectType` property to `MultiSelectListBox`, and the `DataSource` property to `name` (for the name of the pizza). The code to run when the form loads is:

```
Sub Form1_Load(sender As Object, e As EventArgs)

    Dim ds As DataSet = DataLayer.GetData()

    Pizzas.DataSource = ds.Tables("pizza").DefaultView
    Pizzas.DataBind()

End Sub
```

The following screenshot shows how the resulting page is displayed by the Nokia emulator:



This example shows how developing mobile applications with Web Matrix is just as easy as standard Web applications development.

### ***Internet Explorer Web Controls***

The Internet Explorer Web Controls are a set of server controls that provide rich DHTML support for use in Internet Explorer (version 5.5 and above). These can be accessed by customizing the Web Matrix Toolbox. You can then use them as you would any other control; by dragging them onto the design surface. Of the four controls supplied in the IE set, only the TabStrip has designer support – the rest must be customized in either HTML or Code view.

### ***Building and Use a Custom Control***

Building custom server controls can easily be done using Web Matrix, since such controls are just classes. However, Web Matrix doesn't support the automatic building of class files, or some of the advanced features that you might want to use when creating custom controls, such as the ability to embed resources such as a bitmap.

Despite these limitations, and the fact that Web Matrix is designed primarily for editing ASP.NET pages, it's still a fine choice for creating custom controls. Here you can see a simple site content control I've created (that supports templating) by adding a class and using the code editor:

```
using System;
using System.Web;
using System.Web.UI;
using System.Collections;
using System.Web.UI.WebControls;
using System.ComponentModel;

[assembly: TagPrefix("ipona.Controls", "ipona")]
[assembly: AssemblyKeyFile("..\\..\\ipona.snk")]

namespace ipona.Controls
{
    [ToolboxData("<{0}:SiteContent runat=\"server\"></{0}:SiteContent>")]
    public class SiteContent : WebControl, INamingContainer
    {
        . . .
    }
}
```

Since this control will be added to the toolbar the control needs a strong name, which needs to be created manually using the sn.exe utility (you can find more information about sn.exe in the .NET Framework SDK documentation). Then we can compile the class manually, and use the Add Local Toolbox Components option on the Tools menu to add this control to the toolbox:



We haven't added any designer support, but we can drag this control onto a page and then add the content via the HTML view:

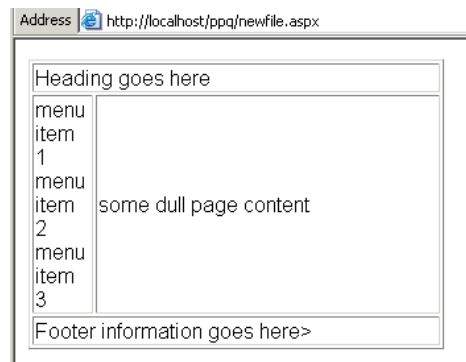
```
<ipona:SiteContent id="SiteContent1" style="WIDTH: 315px; HEIGHT: 151px"
  runat="server">
  <HeadingTemplate>
    Heading goes here
  </HeadingTemplate>

  <MenuTemplate>
    menu item 1<br/>
    menu item 2<br/>
    menu item 3<br/>
  </MenuTemplate>

  <ContentTemplate>
    some dull page content
  </ContentTemplate>

  <FootringTemplate>
    Footer information goes here>
  </FootringTemplate>
</ipona:SiteContent>
```

If we view the page we'll see that the control acts just like any other:



## Part 3 – Configuring and Extending Web Matrix

Web Matrix is designed to be extensible and highly configurable, which allows it to be tailored to suit individual requirements. In this final section, we'll look at the different ways that we can take advantage of this. We'll cover:

- ❑ The Web Matrix configuration settings files
- ❑ The Web Matrix Preferences Dialog
- ❑ Creating and modifying the document templates that appear in the New File dialog
- ❑ Installing and using Add-ins and Code Builders, which can be used to achieve specific tasks
- ❑ Customizing the user interface in general

# The Web Matrix Configuration Files

Just like ASP.NET, Web Matrix obtains most of its run time settings from a configuration file. This file, named `WebMatrix.exe.config`, is located in the Matrix program folder: `\Program Files\Microsoft ASP.NET Web Matrix\version\`.

This folder also contains a file named `ClassBrowser.exe.config`, which contains the run-time settings for the Class Browser tool. The `ClassBrowser.exe.config` file is similar in format to the corresponding section of the `WebMatrix.exe.config` file.

The format and content of Web Matrix's configuration files is likely to change as the product develops, but the current version of `WebMatrix.exe.config` contains the following sections. Note that "saturn" was the early project/development name for Web Matrix, and is still used internally:

```
<configuration>

  <configSections />          - config file sections and corresponding handlers
  <runtime />                 - the assemblies used by Web Matrix
  <appSettings />             - application-specific values for Web Matrix

  <microsoft.saturn>

    <packages />               - the packages that actually implement Web Matrix
    <projects />               - the assemblies to create each project type
    <documentTypes />          - the types of document and template available
    <toolbox />                 - the sections available in the Toolbox
    <addIns />                  - the Add-ins that are available
    <webLinks />                - the list of Help and Community links
    <classView />              - the assemblies and folders in the Classes window
    <dataObjectMappings />     - data type conversion formats for custom objects

  </microsoft.saturn>

</configuration>
```

By editing the contents of these sections of the configuration file, you can change the appearance of Web Matrix, as well as controlling what and where items appear in the IDE. We'll look at some examples later in this section.

Note that Web Matrix caches the actual run-time settings generated from the configuration file in a separate file named `WebMatrix.settings`, which is located in your own "user profile" folder. By default this is:

```
\Documents and Settings
\[username]
\Application Data
\Microsoft Corporation
\ASP.NET Matrix Project
\[version]
\WebMatrix.settings.
```

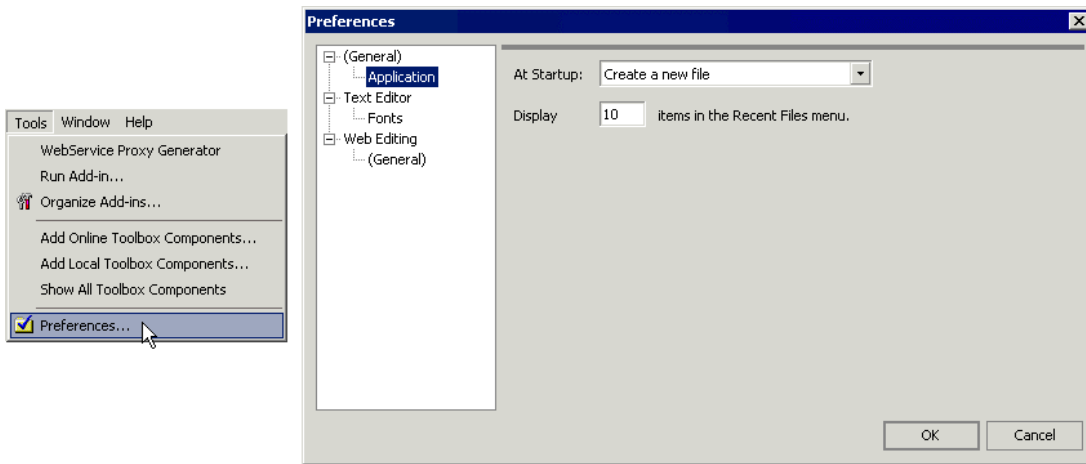
This file is generated automatically each time you close Web Matrix, and then read when you restart Web Matrix.

If you find that your changes to the `WebMatrix.exe.config` file are not picked up when you next start Web Matrix, you should manually delete the `Matrix.settings` file in order to force Web Matrix to re-read the `WebMatrix.exe.config` file and generate a new `WebMatrix.settings` file when you close it down again. Note that this will remove any assemblies that you have added to the Classes window using the Customize dialog. To permanently add assemblies to the Classes window, you must edit the `WebMatrix.exe.config` file as shown in *Customizing the Classes Window*.

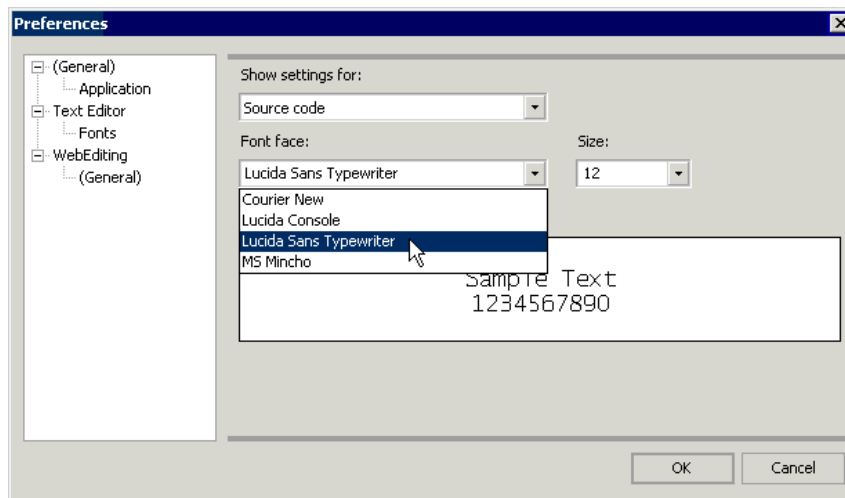
## The Preferences Dialog

You can access the Web Matrix Preferences dialog from the Tools menu. It allows you to modify several features on the IDE. In the (General) section, under the single current entry Application, you can specify what action should take place when Web Matrix starts (show the New File dialog, the Open File dialog, or neither) and control how many entries should appear in the list of recent files on the File menu:



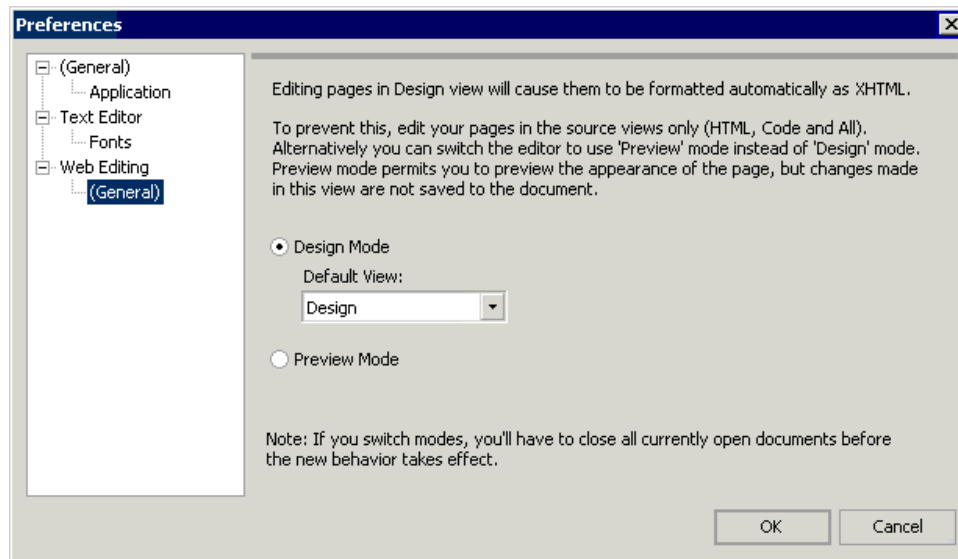


At the moment the Text Editor section just contains the Fonts page. Here you can specify the font to be used for the Edit window in both Source and Code view, and a different font to use for printing the source or code. Only a few fixed-width (mono-space) fonts are available, but you can specify the size and preview the result:



The Web Editing section has a (General) page where you can specify how Web Matrix should treat your code when it's displayed in the edit window. We discussed the two options – Design Mode and Preview Mode – in *The Edit Window* and *Using Preview Mode* in *Part 1*.

In Design Mode, you can edit ASP.NET pages, user controls, and HTML pages in Design view and HTML view (and Code view for ASP.NET pages and user controls). However, in this mode Web Matrix might reformat the code to XHTML when you switch to Design view (so that the resulting rendered appearance can be shown in the edit window). To prevent this, you can switch Web Matrix to Preview Mode, in which case the edit window only has the Source and Preview tabs for ASP.NET pages and user controls, and the HTML and Preview tabs for HTML pages. That way, content can only be edited in Source or HTML view and switching to Preview view does not cause it to be reformatted:



The previous screenshot also shows a drop-down list in which you can specify which view (tab) should be the default when you are using Design Mode. If you prefer you can change the Default View from Design (where the Design view tab is open in the edit window by default) to Source. With this setting, the edit window opens in Code, All, or HTML view by default – depending on the type of file you are editing.

The list of options that are currently available will undoubtedly expand as Web Matrix evolves and develops during its Beta program.

## Creating and Modifying Document Templates

Document templates are used to define the items that appear in the New File dialog, and they contain the content for the pages you create from this dialog. You can edit the document templates that are provided with Web Matrix, and also add your own to suit the kinds of pages and files you regularly create. The template files are stored in a series of subfolders within:

```
\Program Files
 \Microsoft ASP.NET Web Matrix
  \[version]
   \Templates\
```

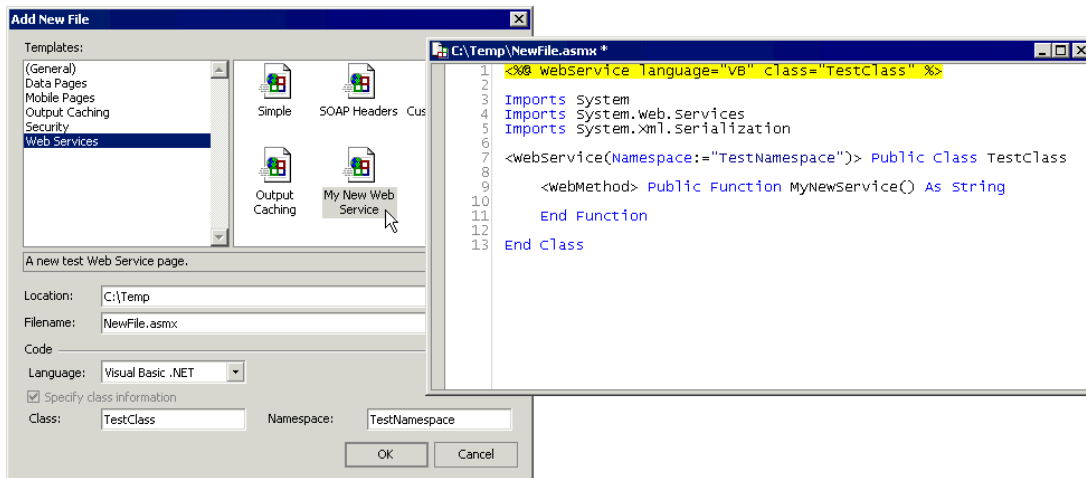
### Editing Document Templates Directly

In the current version of Web Matrix, the only way to modify or add document templates is to edit the appropriate files directly, or place new files into the Web Matrix Templates subfolder hierarchy, and then edit the configuration file.

Let's add a new document template to Web Matrix. First, create a new folder named My New Web Service within Web Matrix's Web Services folder. In this folder create two subfolders, VB and C#, which correspond to the two languages. In each of these folders create a file named NewFile.aspx (the file name can actually be anything you like, though all the supplied templates use the name NewFile). Then it's simply a matter of adding the following to the WebMatrix.exe.config file, within the Web Services Templates part of the <documentTypes> section:

```
<templateDocumentType extension="asmx" templateCategory="Web Services"
  name="My New Web Service"
  createNewDescription="A new test Web Service page."/>
```

The <templateDocumentType> element defines the name of the subfolder, the file extension, the category it will appear in within the New File dialog, and the file description. The following screenshot shows this dialog, together with the file that our new template creates:



The contents of the new file are, of course, governed by the contents of the template. The following listing shows the VB template we placed in the Web Services\My New Web Service\ folder:

```
<%@ WebService language="VB" class="%%ClassName%%" %>

Imports System
Imports System.Web.Services
Imports System.Xml.Serialization

<WebService(Namespace:="%%NamespaceName%%")> Public Class %%ClassName%%

    <WebMethod> Public Function MyNewService() As String

    End Function

End Class
```

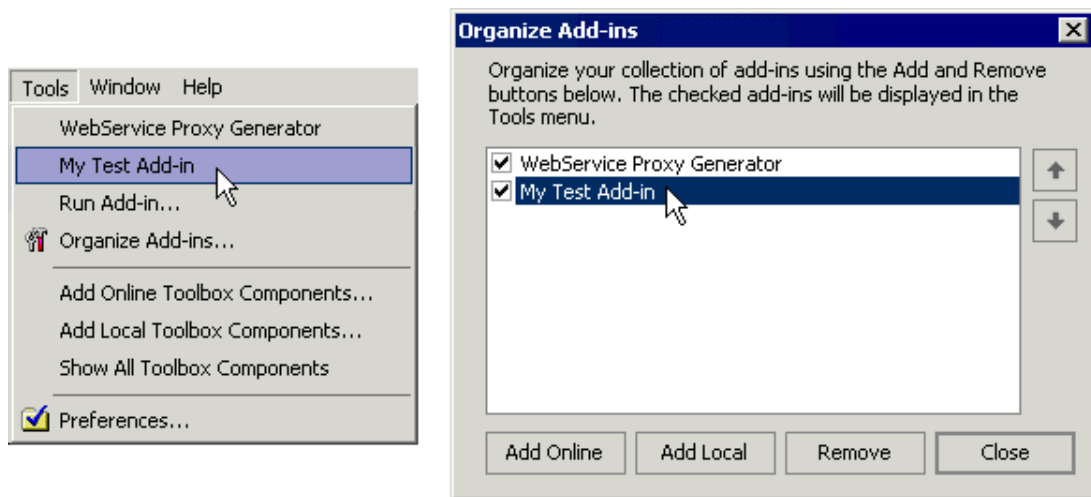
The placeholders for the class and the namespace are enclosed in double percent characters (%%). When the new file is created from the template, the values entered into the Class and Namespace text boxes in the New File dialog are substituted for these placeholders. Of course, the template can contain any code or content that you require – the various Data Pages templates are good examples of this – and you can, if you wish, just edit these rather than adding new ones.

## Installing and Using Add-ins and Code Builders

Web Matrix's extensibility includes both Add-ins and Code Builders, which means that you can supplement the ones provided by default with Web Matrix in order to perform specific tasks that you regularly carry out.

An add-in is a .NET assembly that (usually) has a visual interface, and can be created in Visual Studio .NET using any .NET language. Creating an Add-in isn't a trivial task, but is easy enough if you are competent with Visual Studio .NET. A detailed tutorial, which describes the requirements and the techniques for constructing and installing an add-in, is available from the Web Matrix community Web site.

Add-ins can be installed using the Organize Add-ins dialog, where you use the Add Local button to add an assembly. You can also click Add Online and install any of the add-ins available from the ASP.NET Web Matrix site "gallery" page (which we saw when we looked at installing toolbox components). Once installed, the new add-in will be visible in the Tools menu:



Add-ins can be used to generate code or HTML (perhaps based on a user's input or other environmental features), create or manipulate disk files, run command-line utilities, or even automate other programs outside the Web Matrix environment. Some of the things that you unfortunately cannot do (at least in the current version) are interact with the Web Matrix IDE, or with contents of the edit windows. However, you can copy text to the clipboard, so that users can paste it into an edit window afterwards.

Unlike an add-in, a Code Builder has no visual interface. Instead, it sits in the Code Builders section of the Toolbox, and is used by dragging it onto a page. Like add-ins, Code Builders are created as .NET assemblies, and are installed by placing the assembly in the CodeBuilders subfolder of the main Web Matrix program folder and adding an appropriate `<toolboxItem>` element to the `<toolbox>` section configuration file.

**Detailed instructions for creating Add-ins and Code Builders are outside the scope of this document, but you can find more information and examples on the Microsoft ASP.NET Web Matrix Project home page at <http://www.asp.net/WebMatrix/>.**

## Customizing the Web Matrix Interface

The Classes window and the separate Class Browser tool provide a list of useful Framework classes, from which you can get information about the members of these classes. There are also collapsible folder-based lists of the main ASP.NET classes in the Classes window, which make it easy to quickly find details of these regularly used classes.

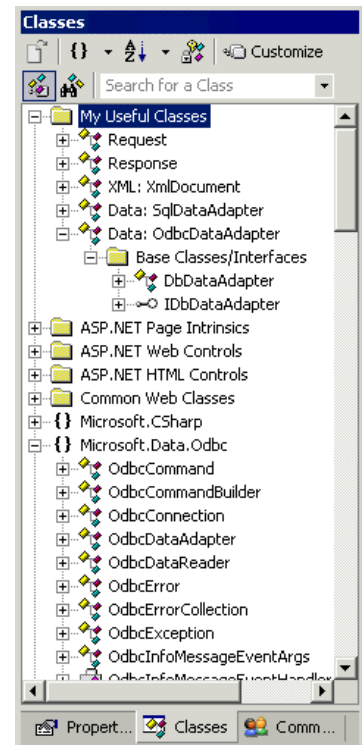
The Customize button at the top right of the Classes window can be used to add assemblies that are already in the GAC (Global Assembly Cache), so that the classes they contain are shown in the list. However, this does not modify the collapsible folders of classes, and the modifications you make using the Customize dialog are lost if you delete the `WebMatrix.settings` file. However, editing the `WebMatrix.exe.config` configuration file provides a way to change these lists of folders, create new folders, as well as permanently adding classes to the list.

## Customizing the Classes Window

In the following screenshot you can see that a new folder named My Useful Classes has been added to the Classes window. This folder has been populated with a few classes from the .NET Framework Class Library. The first four are classes that appear in the other (existing) folders, though a new item has been added to the list (the OdbcDataAdapter class) as well:

Adding a new folder that contains classes from the existing namespaces in the Classes window simply involves adding a new <group> element in the configuration file. To edit the list of classes in an existing folder, you just edit the content of the appropriate <group> element.

Find the <classView> element in the WebMatrix.exe.config configuration file. This element defines which classes that are already qualified in the <runtime> section of the file will appear in the Classes window. In the following listing, you can see the default assemblies, the Microsoft.Data.Odbc assembly we added ourselves, and the custom group we added to create the My Useful Classes folder (and the fact that we used a customized display name in the name attribute):



```
<classView>
  <assembly name="mscorlib" displayName="mscorlib"/>
  <assembly name="System" displayName="System"/>
  <assembly name="System.Data" displayName="System.Data"/>
  <assembly name="System.Drawing" displayName="System.Drawing"/>
  <assembly name="System.Web" displayName="System.Web"/>
  <assembly name="System.Web.Mobile" displayName="System.Web.Mobile"/>
  <assembly name="System.Web.Services" displayName="System.Web.Services"/>
  <assembly name="System.Xml" displayName="System.Xml"/>
  <assembly name="Microsoft.Data.Odbc" displayName="Microsoft.Data.Odbc"/>
  <group name="My Useful Classes">
    <groupItem name="Request" type="System.Web.HttpRequest, System.Web"/>
    <groupItem name="Response" type="System.Web.HttpResponse, System.Web"/>
    <groupItem name="XML: XmlDocument" type="System.Xml.XmlDocument, System.Xml"/>
    <groupItem name="Data: SqlDataAdapter"
      type="System.Data.SqlClient.SqlDataAdapter, System.Data"/>
    <groupItem name="Data: OdbcDataAdapter"
      type="Microsoft.Data.Odbc.OdbcDataAdapter, Microsoft.Data.Odbc"/>
  </group>
  <group name="ASP.NET Page Intrinsic">
    ... etc ...
  </group>
  ...
```

To add a class from an assembly that is not already in the default list, you also have to add that assembly to the <runtime> section of the configuration file. This also ensures that the assembly will be included in the list even if you delete the WebMatrix.settings file at some time in the future (for example when installing your own add-ins or Code Builders). In the next listing, you can see the Microsoft.Data.Odbc assembly we added here so that the classes it contains will permanently appear in the Classes window list:

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <probing privatePath="IDE;Packages;AddIns;Components;CodeBuilders"/>
    <qualifyAssembly partialName="mscorlib" fullName="mscorlib,
      Version=1.0.3300.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
    ... etc ...
    <qualifyAssembly partialName="Microsoft.Data.Odbc" fullName="Microsoft.Data.Odbc,
      Version=1.0.3300.0, Culture=neutral,
```

```

        PublicKeyToken=b77a5c561934e089" />
    </assemblyBinding>
</runtime>

```

*The `fullName` attributes of these elements should be all on one line, and not wrapped as we've had to do in the listing here. This also applies to the `type` attribute of the `<groupItem>` elements we showed in the previous listing.*

To get the information you require to add an assembly to the "qualified" list, you can open the assembly in the `ildasm.exe` tool that is provided with the .NET Framework SDK (in the `Framework SDK\Bin` folder). Once the assembly is opened you can double-click the MANIFEST entry to see the version and public key token values. The `Microsoft.Data.Odbc` assembly that we added implements the .NET ODBC Provider, which is not included in the default installation of the .NET Framework SDK. You can obtain it from the Microsoft Data Access Web site at <http://www.microsoft.com/data/>. By default, it installs the assembly that implements the ODBC provider in:

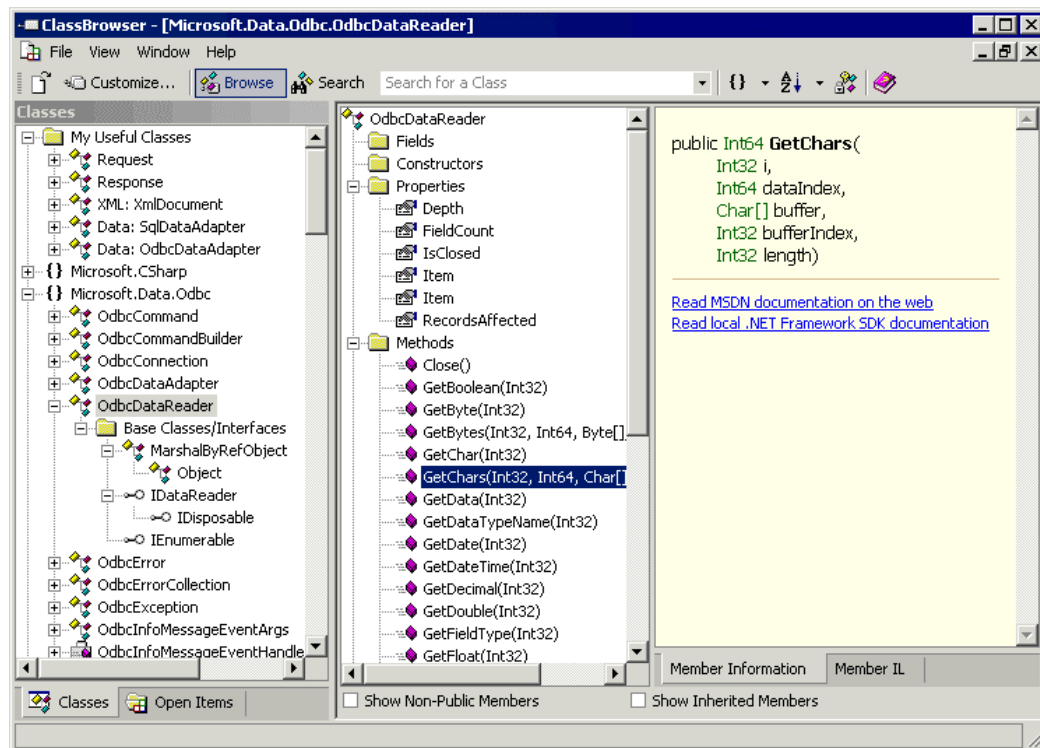
```

\Program Files
\Microsoft.NET
\Odbc.Net
\Microsoft.Data.Odbc.dll

```

## Customizing the Class Browser Assembly List

It's also possible to edit the list of classes that are displayed in the Class Browser tool, and add folders to the list just as we did for the Classes window in the Web Matrix IDE. The `ClassBrowser.exe.config` file contains the same `<classView>` and `<runtime>` elements as the Web Matrix configuration file, so the technique is the same. In the following screenshot, you can see that the My Useful Classes folder has been added to the Class Browser simply by copying it from the `<classView>` section of the `WebMatrix.exe.config` file:



The Class Browser stores its current settings in your Documents and Settings folder in the same way as Web Matrix does. As with Web Matrix, if your changes to the `ClassBrowser.exe.config` file are not picked up when you next open the Class Browser, you should manually delete the `Classbrowser.settings` file to force a re-read of the `ClassBrowser.exe.config` file and the generation of a new `Classbrowser.settings` file when you next close the Class Browser.

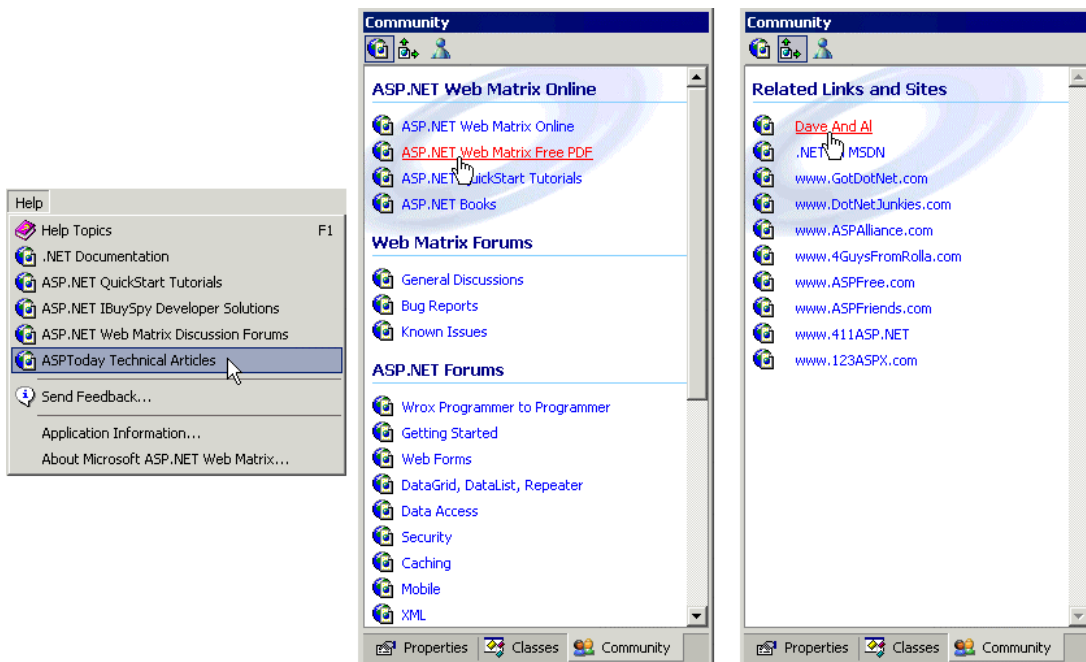
## Customizing Help and Resource Links

Web Matrix provides links to useful resources in several places within the IDE. There are links to the Web Matrix team so that you can send e-mail, links to get help on the Help menu, and a host of links in the Community window. All these can be changed using the `WebMatrix.exe.config` file.

The <webLinks> section of the configuration file contains a list of <webLink> elements. The name attribute for each <webLink> element defines where the link is used. The options are:

- ☐ Help – the link appears in the main Help menu
- ☐ CommunityLink – the link appears in the top section of the first page in the Community window under the ASP.NET Web Matrix Online heading
- ☐ MxForumLink – the link appears in the first page in the Community window under the Web Matrix Forums heading
- ☐ ForumLink – the link appears in the top section of the first page in the Community window under the ASP.NET Forums heading
- ☐ NewsgroupLink – the link appears under the Newsgroups heading in the first page of the Community window
- ☐ ListservLink – the link appears under the Listservs heading in the first page of the Community window
- ☐ RelatedLink – the link appears under the Related Links and Sites heading in the second page of the Community window

You can see the Help menu and some of the headings listed above in the following screenshot, which shows that the default list has been edited and a few of our own links have been added:



Each link type name (such as Help) has a number appended to it when used in the name attribute of the <webLink> element, so the Help menu links shown in the previous screenshot are created by the elements shown in the next listing – we added the last one ourselves:

```
<webLinks>
  <webLink name="Help0" title=".NET Documentation"
    url="http://msdn.microsoft.com/net"/>
  <webLink name="Help1" title="ASP.NET QuickStart Tutorials"
    url="http://www.asp.net/Tools/redirect.aspx?path=quickstart"/>
  <webLink name="Help2" title="ASP.NET IBuySpy Developer Solutions"
    url="http://www.asp.net/Tools/redirect.aspx?path=ibuyspy"/>
  <webLink name="Help3" title="ASP.NET Web Matrix Discussion Forums"
    url="http://www.asp.net/Forums/ShowForumGroup.aspx?tabindex=1&ForumGroupID=4"/>
  <webLink name="Help4" title="ASPToday Technical Articles"
    url="http://www.asptoday.com"/>
  ...
```

Likewise, the extra links in the two pages of the Community window are created with additional <webLink> elements, which we also inserted into the <webLinks> section:

```

...
<webLink name="CommunityLink1" title="ASP.NET Web Matrix Free PDF"
url="http://www.asp.net/WebMatrix/download/pdf"/>
...
<webLink name="ForumLink0" title="Wrox Programmer to Programmer"
url="news://p2p.wrox.com/aspnet"/>
...
<webLink name="NewsgroupLink0" title="news.wrox.com"
url="news://news.wrox.com/aspnet"/>
...
<webLink name="ListservLink0" title="ASP.NET Articles"
url="news://ls.p2p.wrox.com/aspnetnews"/>
...
<webLink name="RelatedLink0" title="Dave And Al"
url="http://www.daveandal.com/" />
...
</webLinks>

```

This allows you to insert your favorite links, as well as allowing you to change the order in which they are displayed.

## Summary

As you can see – even from the very swift tour that we've provided in this document – ASP.NET Web Matrix is already a powerful, intuitive, and extremely useful tool for creating Web sites and Web pages, User Controls, Web Services, and many other types of ASP.NET-related project files. It combines the ease of use of tools like Visual Studio .NET with the simplicity of files created with a text editor or other third-party tools. By avoiding the code-behind approach of Visual Studio .NET, it also produces ASP.NET pages that are generally easier to debug, modify, and extend in the future without requiring the original development tool to be used.

In this document, we looked at the fundamental differences between Web Matrix and Visual Studio .NET, and then went on to tour the IDE itself. We explained the features provided, the templates and wizards that are included, and the way that you can get help and support directly from within the IDE. Remember that Web Matrix is a "community project" that will evolve over time in line with feature requests and feedback from users. The links that are provided to other resources help you to interact with the community at large (outside Microsoft) that has consistently driven the adoption of ASP and ASP.NET.

We went on to put Web Matrix to work and built an example application that demonstrates many of its features. While our application is by no means a real-world commercial application, it did allow us to show several types of files, wizards, and other features that are part of Web Matrix. It also indicates just how much time and effort Web Matrix can save you when you are working with ASP.NET.

Finally, we looked at how you can configure and extend Web Matrix to suit your own requirements. This includes setting preferences as well as modifying the user interface content and layout. We also indicated that you could build your own add-ins and other tools that directly integrate with Web Matrix. There will be an increasing number of ready-built tools and add-ins available as Web Matrix matures and the user community grows. The focus for all this is the Microsoft ASP.NET Web Matrix site at <http://www.asp.net/WebMatrix>.

Bear in mind that this document looks at the Beta 1 version of Web Matrix, released in the summer of 2002. There is no fixed timetable for a release of the "final" product at the moment – in fact there may never be a "final" version. Only you, as a member of the user community, can help to influence and decide the ultimate fate of Web Matrix. Use it and enjoy it.