1

A Fast Track Guide to ASP.NET

Microsoft's .NET technology has attracted a great deal of press since Beta 1 was first released to the world. Since then, mailing lists, newsgroups, and web sites have sprung up containing a mixture of code samples, applications, and articles of various forms. Even if you're not a programmer using existing ASP technology, it's a good bet that you've at least heard of .NET, even if you aren't quite sure what it involves. After all, there's so much information about .NET, that it's sometimes hard to filter out what you need from what's available. With new languages, new designers, and new ways of programming, you might wonder exactly what you need to write ASP.NET applications.

That's where this chapter comes in, because we are going to explain exactly what is required, and how we go about using it. The aim is to get you up and running, able to write simple ASP.NET pages as quickly as possible, and give you a solid grounding in the basics of the new framework. This will not only benefit existing ASP programmers, but also people who haven't used ASP, including Visual Basic programmers who need to write web applications. ASP.NET makes the whole job much easier whatever your skill set.

So, in particular we are going to be looking at:

- ❑ Installing and testing ASP.NET
- ❑ The benefits of the new technology
- ❑ The basic differences between ASP and ASP.NET
- ❑ The new programming model
- ❑ The rich hierarchy of server controls

We start with the simple discussion of why ASP.NET has come about.

Evolution or Revolution?

As developers, we are all used to the evolutionary cycle of software product releases, where each new release adds a few features and cures a bunch of bugs. Server–side web technology has followed this pattern, with products such as dbWeb and the IDC rapidly settling into the Active Server Pages we know and love today. ASP 1.0 was released in 1996, and although it has gone through a further two releases, it hasn't really changed that much – until now. Be prepared to throw away many of those ingrained ASP programming habits, as you've an interesting ride ahead.

ASP.NET is where the revolution begins, because it is radically different from previous versions. Its first appearance into the world was at the Wrox Conference in Washington D.C. back in 1999, where impromptu applause showed how much the audience liked the product. Then in July 2000, ASP.NET received its first public release at PDC, where around 6,000 developers were bombarded with nothing but .NET. As a consequence, they spent most of the week looking like rabbits in headlights – rather dazed and confused with all they had to take in. .NET isn't particularly difficult to understand, but ASP.NET is very different from what we are used to.

That's really the whole crux of the matter. ASP.NET is just a part of the whole .NET framework, but to use ASP.NET effectively you have to understand the underlying architecture. In the next chapter we'll outline this new architecture and the benefits it brings, but for now we need to look at ASP.NET.

Getting Started with ASP.NET

The change to ASP.NET may seem daunting to some, but in the immortal words of Douglas Adams: **don't panic!** Even though there's been a radical change, the basics of ASP.NET are easy to grasp, especially if you've only ever programmed in Visual Basic before. Another important point to highlight is that ASP.NET sits alongside ASP – it doesn't touch existing ASP applications at all. Therefore we don't have to worry about anything that we've previously done suddenly stopping working.

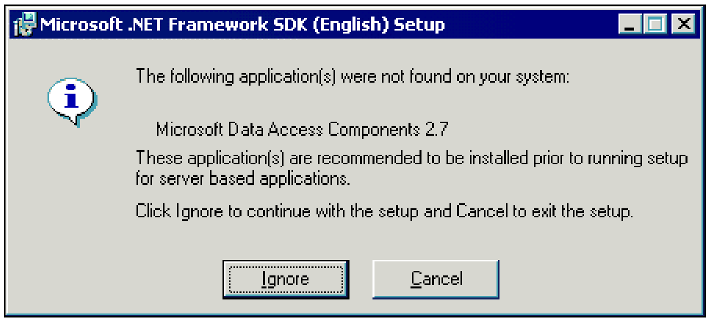
Unlike Beta 2 where there were two versions of ASP.NET, the release version comes in a single version, containing all features.

ASP.NET is supported on Windows 2000 (Professional and Server versions), Windows XP, and will be included in Windows .NET Server. It is not supported for Windows NT or the Windows 9x platforms. You can install Visual Studio .NET on these platforms and remotely use ASP.NET on the supported platforms. ASP.NET can be obtained from Microsoft, at http://www.Microsoft.com/net, http://www.asp.net/ or http://www.gotdotnet.com/, and is also part of the MSDN Subscription service.

Installing .NET

Installation is extremely simple, consisting of a single executable. This installs the framework, including ASP.NET, and includes options for the samples and documentation. During installation you may be asked to update the Microsoft Windows Installer components, and if so, you should click the Yes button to update them. This update is required for the .NET SDK installation.

You may also see the following dialog:



The following application(s) were not found on your system:

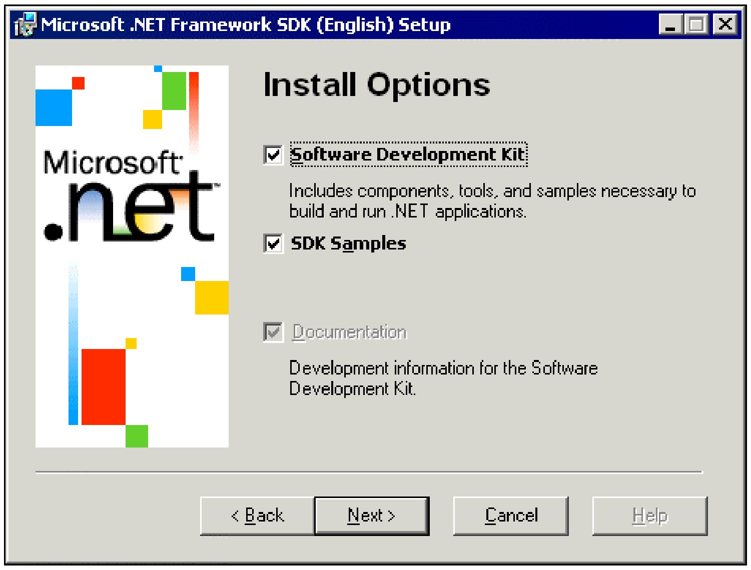
Microsoft Data Access Components 2.7

These application(s) are recommended to be installed prior to running setup for server based applications.

Click Ignore to continue with the setup and Cancel to exit the setup.

This indicates that MDAC 2.7 is not installed on your system. You can press the Ignore button to continue with the setup process – MDAC 2.7 isn't required for .NET, although it is recommended if you use any of the data features that interoperate with ADO.

Once the Installation Wizard starts you'll have the usual license screen followed by the options screen:



Install Options

☑ Software Development Kit

Includes components, tools, and samples necessary to build and run .NET applications.

☑ SDK Samples

☑ Documentation

Development information for the Software Development Kit.

This gives you the options of installing the required components, tools and samples, as well as the SDK samples. You should leave all options ticked to ensure that everything is installed. The distributable version of the .NET framework is around 18Mb, and doesn't contain samples or documentation.

As part of the samples, a named instance of the Microsoft Data Engine (MSDE) is installed containing sample databases.

Configuring the Samples

The installation routine creates a folder called Microsoft .NET Framework SDK containing an HTML page titled Samples and QuickStart Tutorials. From this page you should follow the steps outlined:

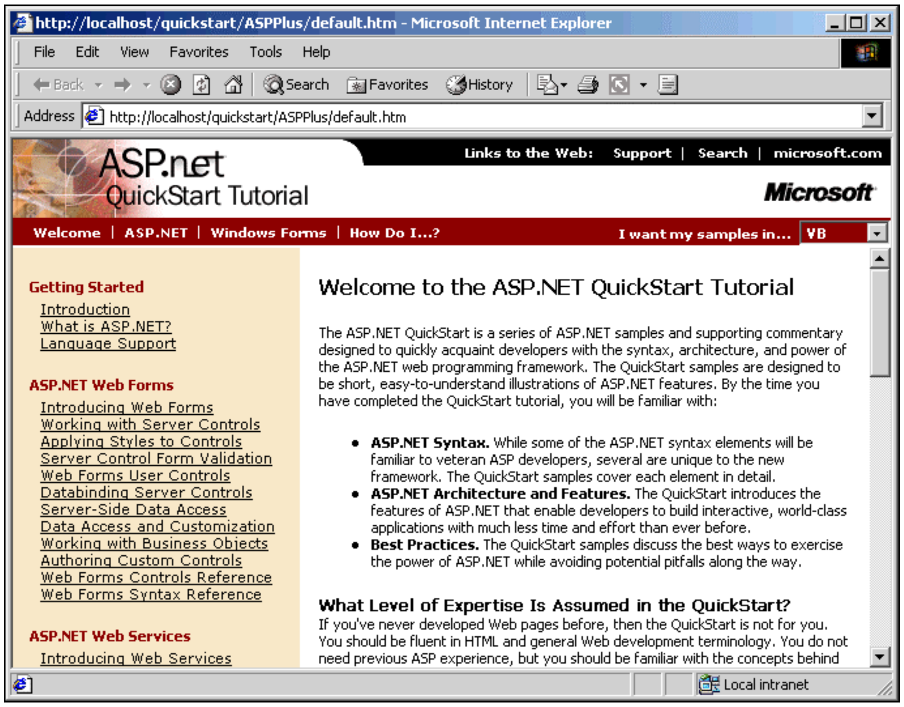
Step 1: Install the .NET Framework Samples Database. Click this link and select Run this program from its current location to run the samples database installation routine. If you receive a Security Warning dialog you can select Yes to allow the program to run. At this point the program checks for MSDE, installing it if it isn't already installed, and then installs the sample databases.

Step 2: Set up the QuickStarts. Click this link and select Run this program from its current location to configure IIS and perform other installation routines. You may also receive another Security Warning dialog when you run this program, and you can select Yes to allow the program to run.

At this point the samples are installed, and you have the option to Launch them. You can also launch the samples by navigating to the Microsoft .NET Framework SDK menu (installed under the Programs) and selecting Samples and QuickStart Tutorials.

Running the Samples

From the main QuickStart page you should select Start the ASP.NET QuickStart Tutorial, where you will be presented with the following screen:


http://localhost/quickstart/ASPPlus/default.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/quickstart/ASPPlus/default.htm
ASP.net QuickStart Tutorial
Links to the Web: Support | Search | microsoft.com
Microsoft
Welcome | ASP.NET | Windows Forms | How Do I...?
I want my samples in... VB
Local intranet

Getting Started
- Introduction
- What is ASP.NET?
- Language Support

ASP.NET Web Forms
- Introducing Web Forms
- Working with Server Controls
- Applying Styles to Controls
- Server Control Form Validation
- Web Forms User Controls
- Databinding Server Controls
- Server-Side Data Access
- Data Access and Customization
- Working with Business Objects
- Authoring Custom Controls
- Web Forms Controls Reference
- Web Forms Syntax Reference

ASP.NET Web Services
- Introducing Web Services

Welcome to the ASP.NET QuickStart Tutorial

The ASP.NET QuickStart is a series of ASP.NET samples and supporting commentary designed to quickly acquaint developers with the syntax, architecture, and power of the ASP.NET web programming framework. The QuickStart samples are designed to be short, easy-to-understand illustrations of ASP.NET features. By the time you have completed the QuickStart tutorial, you will be familiar with:

- **ASP.NET Syntax.** While some of the ASP.NET syntax elements will be familiar to veteran ASP developers, several are unique to the new framework. The QuickStart samples cover each element in detail.
- **ASP.NET Architecture and Features.** The QuickStart introduces the features of ASP.NET that enable developers to build interactive, world-class applications with much less time and effort than ever before.
- **Best Practices.** The QuickStart samples discuss the best ways to exercise the power of ASP.NET while avoiding potential pitfalls along the way.

What Level of Expertise Is Assumed in the QuickStart?
If you've never developed Web pages before, then the QuickStart is not for you. You should be fluent in HTML and general Web development terminology. You do not need previous ASP experience, but you should be familiar with the concepts behind

The left-hand portion of the screen shows the samples broken into their groups, which are:

Sample Group	**Consists of …**
Getting Started	Introduction to ASP.NET and the .NET languages.
ASP.NET Web Forms	The basics of ASP.NET page design, including use of server controls, databases and business objects.
ASP.NET Web Services	How to create and use Web Services.
ASP.NET Web Applications	What defines an ASP.NET application, and how the global files are used.
Cache Services	The new cache features, allowing pages or data to be cached to improve performance.
Configuration	The new XML-based application configuration.
Deployment	A description of how applications are deployed.
Security	An examination of the authentication and authorization features in the .NET framework.
Localization	Examples of how internationalization can be achieved.
Tracing	How the new tracing features of ASP.NET bring increased developer productivity.
Debugging	How to use the new visual debugger.
Performance	Overview and tips and tricks on improving performance.
ASP to ASP.NET Migration	Examples showing how to migrate existing applications.
Sample Applications	Some sample applications, described below.

We'll see examples of these topics throughout the book.

The right-hand side of the screen will show the samples, including descriptions and sourcecode. The sourcecode for all of the samples is available in Visual Basic, C#, and JScript. The use of these languages is discussed later in the chapter.

The Sample Applications

The sample applications should give you some good ideas of what can be achieved with ASP.NET, as well as showing how it can be achieved and some best practices for writing applications.

- ❑ **A Personalized Portal** is a sample portal application, allowing user login, content delivery, user preferences, configuration, and so on. It's an extremely good example of the use of User Controls, which are reusable ASP.NET pages.
- ❑ **An E–Commerce Storefront** is a small electronic–commerce site, based around a simple grocery store. It shows some good uses of data binding and templating, and how a shopping basket system could be implemented.

- **A Class Browser Application** shows how we can browse through the hierarchy of classes and objects. Not only is this useful from a learning point of view, but it also shows how the classes are queried by run–time code. This is one of the great new features of the framework, and is explained in more detail in the next chapter.
- **IBuySpy.com** is another electronic–commerce site, showing more features than the other sample store. It contains user logins, shopping baskets, and so on.

Additional Samples

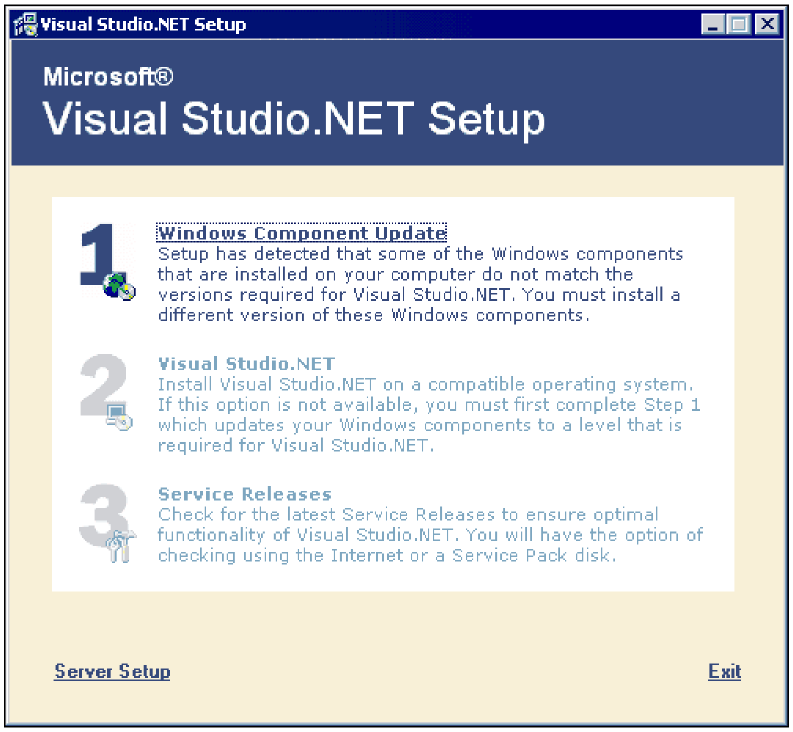
The above list of samples describes just the ones that are installed by the SDK, but there are plenty of others available, such as a .NET version of the Duwamish site. All of the code for the samples in the book is available from the Wrox Press web site (at www.wrox.com). Microsoft has three additional sites where information and samples can be obtained:

- **www.asp.net** is the central site for downloads and links.
- **www.ibuyspy.com** is the IBuySpy application online. This code runs online as well as being available as a download (in VB.NET and C#). This site also contains links to a portal based version of IBuySpy, allowing user customization, and a news based version, aimed at content delivery.
- **www.gotdotnet.com** is a community site for all .NET developers. It's full of links and samples by both Microsoft and third parties. This site also has a list of ASP.NET hosting companies. There are also plenty of third party sites, and since this list may change, your best bet is to go to www.gotdotnet.com and follow the links page.

Visual Studio .NET

Although this book is primarily aimed at ASP.NET, it is important that we mention Visual Studio .NET as well. The first thing to make clear is that Visual Studio .NET isn't required to write ASP.NET applications, but it does provides an extremely rich design environment. It provides features such as drag and drop for controls, automatic grid and list support, integrated debugging, Intellisense, and so on.

The installation of Visual Studio .NET comprises several steps:



Windows Component Update
Setup has detected that some of the Windows components that are installed on your computer do not match the versions required for Visual Studio.NET. You must install a different version of these Windows components.

Visual Studio.NET
Install Visual Studio.NET on a compatible operating system. If this option is not available, you must first complete Step 1 which updates your Windows components to a level that is required for Visual Studio.NET.

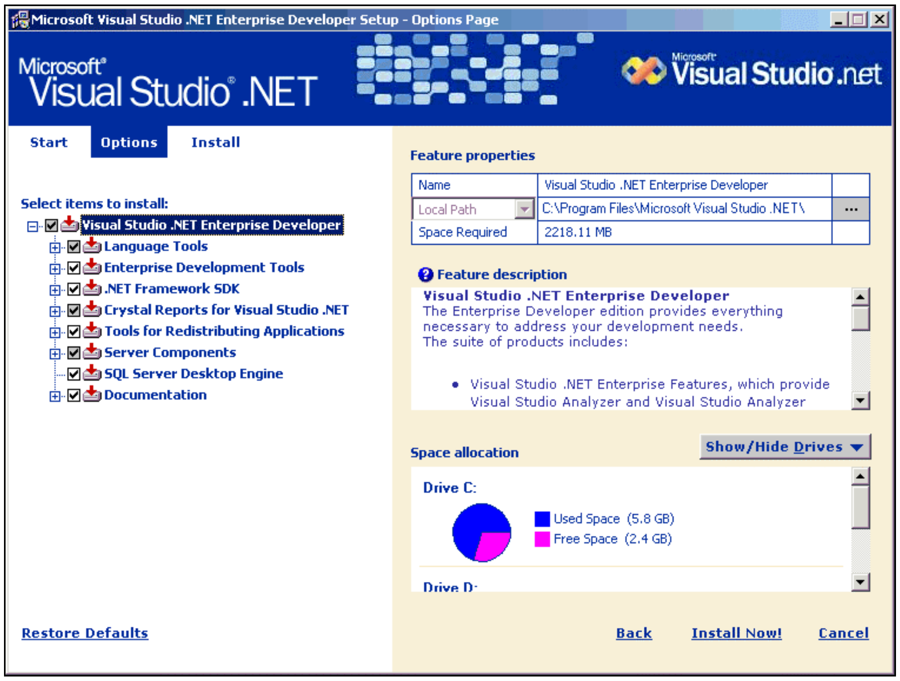
Service Releases
Check for the latest Service Releases to ensure optimal functionality of Visual Studio.NET. You will have the option of checking using the Internet or a Service Pack disk.

The Component Update installs the following:

- ❑ Windows 2000 Service Pack 2, if installing on Windows 2000 (this requires a reboot)
- ❑ Microsoft Windows Installer 2.0
- ❑ Microsoft FrontPage 2000 Web Extensions Client
- ❑ Setup Runtime Files
- ❑ Microsoft Internet Explorer 6.0 and Internet Tools (this requires a reboot)
- ❑ Microsoft Data Access Components 2.7
- ❑ Microsoft .NET Framework

The Component Update install allows you to enter a login name and password to be used during the reboots, so that the entire installation can take place without user interaction.

The Visual Studio .NET install offers a similar setup to previous versions:


Microsoft Visual Studio .NET Enterprise Developer Setup - Options Page
Microsoft Visual Studio .NET
Microsoft Visual Studio.net
Start
Options
Install

Select items to install:

- Visual Studio .NET Enterprise Developer
  - Language Tools
  - Enterprise Development Tools
  - .NET Framework SDK
  - Crystal Reports for Visual Studio .NET
  - Tools for Redistributing Applications
  - Server Components
  - SQL Server Desktop Engine
  - Documentation

Feature properties

Name	Visual Studio .NET Enterprise Developer
Local Path	C:\Program Files\Microsoft Visual Studio .NET\
Space Required	2218.11 MB

Feature description

Visual Studio .NET Enterprise Developer
The Enterprise Developer edition provides everything necessary to address your development needs.
The suite of products includes:

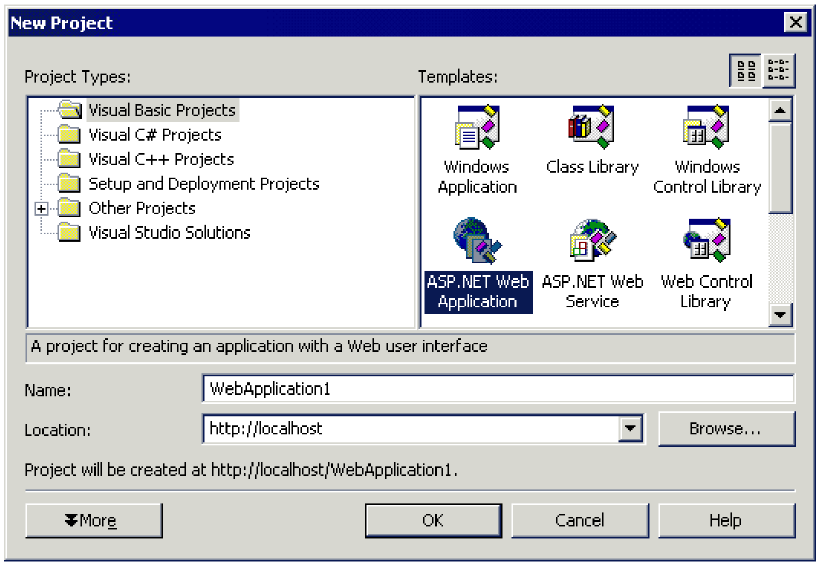
- Visual Studio .NET Enterprise Features, which provide Visual Studio Analyzer and Visual Studio Analyzer

Space allocation

Drive C:
Used Space (5.8 GB)
Free Space (2.4 GB)
Drive D:

Once this step is finished, you have the option of a check for Service Releases, to allow product updates to be automatically downloaded for you.

If you've used previous version of Visual Studio, you may think that the installed menu items are rather sparse, since you only get two or three items (depending upon your installation options). What's noticeable is that the two main items are Microsoft Visual Studio .NET 7.0 and Microsoft Visual Studio .NET Documentation. Because the underlying .NET architecture changes the way languages are used, Visual Studio .NET has been built to take this into account. So, no longer do you pick your language and then run the tool associated with that language. Now you just start Visual Studio .NET and then decide in which language you wish to write, and the type of application to create:



New Project

Project Types:
- Visual Basic Projects
- Visual C# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects
- Visual Studio Solutions

Templates:
- Windows Application
- Class Library
- Windows Control Library
- ASP.NET Web Application
- ASP.NET Web Service
- Web Control Library

A project for creating an application with a Web user interface

Name: WebApplication1

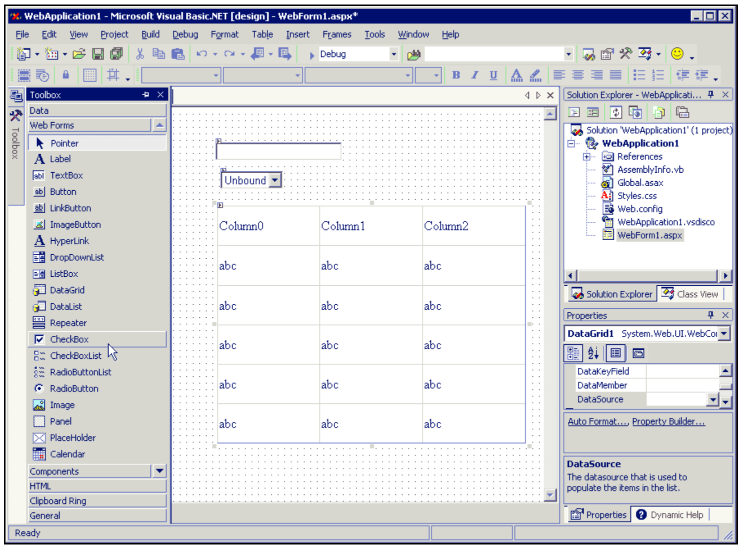
Location: http://localhost

Project will be created at http://localhost/WebApplication1.

What's great about this, is that the development environment is the same, whatever the language and application. This dramatically reduces training time, as you don't have to learn a different tool to do something differently.

Creating ASP.NET Applications in Visual Studio .NET

When using Visual Studio .NET, you select ASP.NET Web Application from the New Project dialog (shown above), and this creates the named web site and creates some default pages. From that point onwards you just use the design environment to drag controls onto the design grid:



WebApplication1 - Microsoft Visual Basic.NET [design] - WebForm1.aspx*

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

Solution 'WebApplication1' (1 project)
- WebApplication1
  - References
  - AssemblyInfo.vb
  - Global.asax
  - Styles.css
  - Web.config
  - WebApplication1.vsdisco
  - WebForm1.aspx

Properties: DataGrid1 System.Web.UI.WebCo

DataKeyField
DataMember
DataSource

DataSource
The datasource that is used to populate the items in the list.

You can then use View Code (or the more familiar double-click on a control) to see the code for the web page you are creating.

We're not going to go into any more detail on using Visual Studio .NET, as it's too big a topic and really is outside the scope of this book. What we really want to concentrate on is ASP.NET itself.

Other Installs

There are several other related technologies that are not included as part of .NET, but which you might find useful. These are:

- ODBC .NET Data Provider, which provides access to native ODBC drivers.
- Mobile Internet Toolkit, to allow development of sites that support mobile devices, such as phones and PDAs.
- Internet Explorer Web Controls, provide a set of client controls (such as a TreeView and Tab Control) for use in Internet Explorer.
- Internet Explorer Web Services Behavior.

Not all of these are running to the same timeframe as the .NET SDK, but they should all be available from http://www.Microsoft.com/downloads or from MSDN.

How is ASP.NET Different from ASP?

This question can be answered in one word – very. ASP.NET is not just a new version, but a whole new idea and way of programming web applications. New features weren't retrofitted into ASP to give us a new version – ASP.NET has been written from the ground up to provide the best possible application framework. This has meant that, in many areas, compatibility with ASP has been broken, but in the long term this is a good thing. It means that ASP.NET provides a much stronger platform for developing applications, and gives many more benefits.

If you're worried about the compatibility issue, then remember we mentioned earlier that ASP.NET runs alongside ASP. Even though there are many differences between the two, installing ASP.NET won't break existing applications. That's because your existing ASP pages are still processed by the same mechanism as before, and the new framework processes ASP.NET pages. This is achieved by ASP.NET pages having a new file extension (`.aspx`), meaning they are not processed in the same way as ASP pages.

Compatibility and migration issues are covered in Chapter 23.

Why Do We Need a New Version?

ASP has achieved enormous success as a way of developing web sites, so why is a new version needed? Simply put, ASP hasn't evolved to take into account the way it's now being used. Although designed with great scope and flexibility, I don't think even its authors could have seen how it would become the cornerstone of many applications. Like a tempestuous Hollywood starlet, its rapid rise to fame has led to problems:

- ASP is a scripted language, relying mainly on VBScript and JScript. Other languages are available if we install an interpreter, but it's still interpreted. The two disadvantages of interpreted languages are the lack of strong types (as supported by typed languages such as Visual Basic and C/C++), and the lack of a compiled environment. ASP does cache code, but it's still interpreted, and this inevitably leads to performance and scalability problems.
- ASP doesn't provide an inherent structure for applications. In the days of static web pages, we used to see small, focused source files. With the dynamic concept of ASP, it was possible to build code into the web page, again leading to problems. There's the eternal worry of mixing code and content, which can be a problem if you have a mixed team, with certain people designing the HTML and the interface, with different people doing the coding. Having two sets of people working on the same files is asking for trouble. Another problem was the ability to make the code complex, leading to larger source files. Include files allow a certain amount of structure and code reuse, but it was never really a great solution.
- We have to write code in ASP to do most things, no matter how simple. For example, consider the task of validating form fields. Just to ensure that values are entered into a field requires code. Other areas such as caching content, maintaining form state, and so on, all require code. Even adding new HTML controls requires writing the raw HTML to the page.
- The world of browser compatibility has morphed into device compatibility. While the majority of web access still takes place from a PC and browser, how long will that remain the case? Mobile devices are becoming more prevalent, and more powerful, leading to more problems designing sites. If you want your web site to obtain maximum reach you need to contend with these devices, and this means writing code to detect the device and render the appropriate content.
- Standards compatibility also plays a big part in web development. XHTML is becoming more widely accepted, XML and XSL/T are both now widely used, and talking to mobile devices might also mean support for WML. Support for these standards mean that our ASP applications not only have to work with existing standards, but also be easily upgradeable to support future standards.

These are just some of the problems we will encounter when building ASP applications, but they aren't the only ones. The rapidly changing nature of the Internet often requires rapid changes to applications. For languages that have strong development environments, practices such as componentization, code reuse, rapid development, and so on, are a great boon to a developer, but this sort of support is lacking in ASP. The rise of Business-to-Business applications, and peer-to-peer data sharing also brings great challenges to the developer.

ASP.NET was written from the ground up to meet these needs. Not only does it answer many of the questions posed by the existing development environment, but also provides great extensibility, and brings great tool support. At its minimum, all you require is the ASP.NET redistributable, which is freely available, and you can continue to use your favorite editor of choice (come on, admit it – it's Notepad). This gives us access to everything possible with ASP.NET, including multi-language support. For a richer environment you can use Visual Studio .NET, where you get the drag and drop support, colored code (more useful than you'd think), context sensitive help and tooltips, and all of the usual great editing features that Visual Studio has brought in the past.

Benefits of ASP.NET

From the above discussion of the problems with ASP it would be easy to say that ASP.NET solves those problems, and while that is so, there's a lot more to it than that. To understand what's been done, have a look at four of the main goals of ASP.NET:

- Make code cleaner
- Improve deployment, scalability, security, and reliability
- Provide better support for different browsers and devices
- Enable a new breed of web applications

You may not see some of this support directly, as the Common Language Runtime (CLR) handles much of it. This is discussed in detail in the next chapter, but for now we are going to concentrate on how ASP.NET improves our lives.

Multiple Languages

ASP has been limited to scripting engines, notably VBScript and JScript. The .NET framework inherently supports multiple languages, so we can use whichever we feel most comfortable with. By default the CLR comes with Visual Basic .NET, C#, and JScript .NET (all compiled), and there are a number of third party languages that we can use, such as Perl, COBOL, and many others. Additionally, Visual Studio .NET adds support for Visual C++, and an implementation of Java (called J#) is also available for download from Microsoft. Because this language support is part of the framework, it really doesn't matter what language you, or others in your team, use. Obviously, from your point of a view, it's probably best to maintain some degree of compatibility (for maintenance purposes if nothing else), but as far as the framework is concerned, anything goes.

This multiple language support isn't just limited to what's available, but also to how it's used. It's quite possible to write components in one language, and use (or reuse) them from another language. The server based controls are written in C#, but we can quite happily sub-class them from Visual Basic, and then sub-class that control in JScript (or any .NET supported language).

The framework is covered in more detail in the next chapter, while Chapter 3 delves into the languages themselves in more detail.

Server Processing

If you've done some Visual Basic programming, then you'll find the switch to the new ASP.NET Server Controls fairly painless, but they might cause some initial confusion if your programming has been limited to ASP. There's no need to worry though, as they are extremely easy to understand and use – it's just that they are very different from ASP.

One of the big problems with ASP is that pages simply define one big function, which started at the top of the page and finished at the bottom. The page content is rendered in the page order, whether it is straight HTML or ASP-generated HTML. Therefore, our logic was dependent upon its position in the page, and there's no way to target HTML controls except by rendering them as part of the stream. Anything we do requires us to write code, and that includes the output of HTML elements.

ASP.NET solves this problem by introducing a declarative, server-based model for controls. This is where the concept may seem alien to ASP programmers, because the controls are declared on the server, can be programmed against on the server, but can be event driven from the client. This sounds pretty weird, but it's simple to use. All we have to do to turn a normal HTML control into a server control is add `runat="server"` as an attribute. For example:

```
<input id="FirstName" type="text" runat="server">
```

This is a standard HTML control, but the addition of the `runat` attribute allows the control to be programmed against with server-side code. For example, if this control is placed within a form and we submit the form back to the same page, we can do this in our server-side code:

```
Dim PersonFirstName As String

PersonFirstName = FirstName.Text
```

Making a control run on the server allows us to use the `ID` attribute to identify it directly. This allows the code to become more readable, since we don't have to refer to the form contents or copy the contents into variables. It's also more natural to refer to the control directly, which makes developing pages simpler. If you've done any Visual Basic or VBA programming then this won't seem too alien for you.

If you've only ever done scripting in ASP, then this may seem strange, but that's only because it's a different way of working with content to and from the browser. You've probably done database access, so you've used objects, called methods, and set properties, and the ASP.NET Server Controls aren't really any different from this.

The new server processing architecture is covered in Chapter 4.

Web Form Controls

Converting existing HTML controls to server-side ones is simple, but there are still several problems with this approach:

- **Consistency**. We are still stuck with the rather non-intuitive nature of some HTML controls. Why for example, is there an `INPUT` tag for single line text entry, but a `TEXTAREA` tag for multi-line text entry? Surely a single control where we specify the rows and columns makes more sense?
- **User Experience**. How do we easily write sites that render rich content for browsers such as IE, while also preserving compatibility with down level browsers? HTML doesn't have the ability to change its content depending on the browser – we have to write the code for that.
- **Devices**. How do we write sites that cope with devices other than browsers? WAP-Phones, PDAs, and even fridges have browsers nowadays. Like the browser issue, we'd have to manually write code for this.

To alleviate these problems Microsoft has created a set of server controls, identified by the `asp:` prefix. The ASP.NET server controls tackle the above problems by:

- Providing a consistent naming standard. For example, all text entry fields are handled by the `TextBox` control. For the different modes (multi-line, password, etc.) we just specify attributes.
- Providing consistent properties. All server controls use a consistent set of properties, making it easier to remember. For example, the `Text` field of a `TextBox` is more intuitive than a `Value` field.
- Providing a consistent event model. Traditional ASP pages often have large amounts of code handling the posting of data, especially when one page provides multiple commands. With ASP.NET we wire-up controls to event procedures, giving our server-side code more structure.
- Emitting pure HTML, or HTML plus client-side JavaScript. With one minor exception (which is intentional) the server controls emit HTML 3.2 by default, giving great cross-browser compatibility. This can be changed so that by default we target up-level browsers such as IE, where the controls will emit HTML 4.0 and DHTML, providing a richer interface. All the user ever sees is the HTML content, not the server controls.
- Emitting device specific code. Certain controls will emit HTML when requested by a browser, but WML when requested by a WAP phone. The control handles the detection of the device and the generation of the correct markup.

The controls will be covered in detail in later chapters, but let's take a quick look at a simple example to show how these controls work:

```
<html>

<script language="VB" runat="server">

 Public Sub btn_Click(Sender As Object, E As EventArgs)

  ' some code goes here

 End Sub

</script>

<body>

<form runat="server">

 Press the button: <asp:Button runat="server"
             Text="Press Me" OnClick="btn_Click"
             runat="server"/>
</form>

</body>
</html>
```

The server control in this example is a button, added to the page using the `asp:Button` element. There are several things to note about this control:

- ❑ It has the `runat="server"` attribute set, to tell ASP.NET that it should process this control.
- ❑ It uses the `Text` attribute to set the text to be shown on the button. This is consistent with other controls.
- ❑ It uses the `OnClick` attribute to identify the event procedure to be run when the button is clicked. Since this is a server control this event procedure runs on the server.

The event procedure is automatically supplied with two parameters – the control that generated the event, and any additional arguments the procedure requires. Within the event procedure we can access any other server controls, including the contents of input fields submitted during a postback.

HTML Output

In traditional ASP pages, the ASP processor runs server-side code, stripping it out so that only HTML and client-side script is sent to the client. This process is exactly the same for ASP.NET pages (the `<% %>` tags still work), with the server controls being converted to their HTML equivalents. For example, the page code shown above renders the following HTML to the browser:

```
<html>
<body>

<form name="ctrl2" method="post" action="test.aspx" id="ctrl2">
<input type="hidden" name="__VIEWSTATE"
    value="YTB6MTU5NDYxNjE5Ml9fX3g=2dbab7f5" />

 Press the button: <input type="submit" name="ctrl5" value="Press Me" />
</form>

</body>
</html>
```

There are several things to note here:

- ❑ The first is that the `form` has `method`, `action`, and `id` attributes added automatically. We can add these in ourselves (with the exception of the action attribute) if we want to, but it's not necessary.
- ❑ A hidden input field is added, which contains (in a compressed form) the state of the server controls. This is called the ViewState, and is how ASP.NET manages the content of the controls. View State is covered in Chapter 4.
- ❑ The Button is converted into a standard submit button.

So, we can see that even though we have better code on the server, it doesn't affect how the code is presented on the client. It's still standard HTML, with standard forms and elements.

Server Control Hierarchy

The server controls are logically broken down into a set of families:

- **HTML Server Controls**, which are the server equivalents of the HTML elements**.**
- **Web Form Controls**, which map closely to individual HTML elements.
- **List Controls**, which map to groups of HTML elements that produce grids or grid-like layout.
- **Rich Controls**, which produce rich content and encapsulate complex functionality, and will output pure HTML or HTML and script. A good example of this is the Calendar control, which provides the user with a calendar from only one line of code.
- **Validation Controls**, which are non-visible controls, but allow the easy use of both server-side and client-side form validation.
- **Mobile Controls**, which output HTML or WML depending upon the device accessing the page.

Chapters 5 and 6 deal extensively with most of these controls, and Chapter 21 covers the Mobile Controls.

At this early stage in the book, you may not be able to see the implications that these controls have for you, but let's take a couple of common examples. First off, the case of displaying data from a database, perhaps in some form of grid. In ASP, we'd open the Recordset containing the data, and loop through the rows and columns building up an HTML table. We might well have this abstracted into a separate function in an include file, but we still had to write the code. With the ASP.NET DataGrid control, it's the control itself that handles this for us. The list controls (which include the DataGrid) have built in support for extracting data from a data source and creating the HTML for us. For example, consider the following ASP code:

```
<%
 Dim rs
 Dim fld

 Set rs = Server.CreateObject("ADODB.Recordset")

 rs.Open "select * from authors", _
   "Provider=SQLOLEDB; Data Source=.; Initial Catalog=pubs; UID=sa; PWD="

 If Not rs.EOF Then
  Response.Write "<table border='1'><tr>"
  For Each fld In rs.Fields
   Response.Write "<td>" & fld.Name & "</td>"
  Next
  Response.Write "</tr>"

  While Not rs.EOF
   Response.Write "<tr>"
   For Each fld In rs.Fields
    Response.Write "<td>" & fld.Value & "</td>"
   Next
   Response.Write "</tr>"
   rs.MoveNext
  Wend

  Response.Write "</table>"
 End If
%>
```

There's nothing special about this – it just creates an HTML table. Now compare this to the equivalent ASP.NET code using a DataGrid:

```
<%@ Import Namespace="System.Data.SqlClient" %>

<script language="VB" runat="server">

 Sub Page_Load(Sender As Object, E As EventArgs)

  Dim con As New SqlConnection("Data Source=.; " & _
               "Initial Catalog=pubs; UID=sa; PWD=")
  Dim cmd As SqlCommand

  con.Open()
  cmd = New SqlCommand("select * from authors", con)

  DataGrid1.DataSource = cmd.ExecuteReader()
  DataGrid1.DataBind()
  con.Close()
 End Sub

</script>

<asp:DataGrid id="DataGrid1" runat="server"/>
```

Note that you should always close a database connection when you have finished with it. Either call the `Close` method of the `Connection` object, or pass the value `CommandBehavior.CloseConnection` as the parameter to the `ExecuteReader` method. See Chapter 8 for more details.

We can immediately see how there's much less code to write. In fact, all of the code here relates to getting the data from the database and binding it to the grid. There isn't any code to create a table as the DataGrid does this.

Data binding is covered in Chapter 7.

Another great example of the power of controls is the Calendar control, which with one line of code creates a fully functional calendar on our web page:

```
<asp:Calendar runat="server"/>
```

That's it – nothing extra is needed to get it working.

This sort of simplified approach doesn't mean that the controls are simple, just simple to use. The onus on coding has moved from the web page developer to the control developer. There are also plenty of other non-Microsoft controls, either planned or released, covering everything from more advanced grids to TreeViews. Alternatively you can write your own controls. This is covered in Chapter 18.

Language Improvements

One of the greatest new features is that scripting is dead – hooray. This is a slight exaggeration, as what's really dead is the typeless, interpreted nature of these languages. VBScript is no longer supported, and is replaced with full Visual Basic support, while JScript is still supported but has the addition of types. In addition, a new language called C# (pronounced C Sharp) is introduced, with a format similar to C/C++. As ASP.NET is entirely written in C# we can understand that this isn't a minor addition

We look at the detailed improvements in languages in Chapter 3, but for now, all we need to understand is that all languages:

- Support data types.
- Use a common set of data types.
- Are fully compiled.
- Are object oriented, and support inheritance.

What's also important is that the language support is built into the Common Language Runtime (CLR), which provides this common support. This means that things such as inheritance are cross-language, so we can write components in C# and inherit and extend them in Visual Basic. The CLR manages all of this for us, as well as providing cross-language debugging, giving such features as being able to use a debugger to step through Visual Basic code in an ASP.NET page into a C# component.

What's also provided is extensibility, meaning that additional languages can be supported. Microsoft supply VB.NET, JScript, and C# as standard with the .NET SDK, but many other languages are being worked on by third parties.

Code and Content Separation

I think that this is generally an unused feature of web site design, as many sites are created entirely by programmers. In itself this isn't a bad thing, but I think programmers in general don't make great designers, and I count myself firmly in this group. While I'm extremely interested in interface design and usability, I'm not particularly good at it. ASP tended to build on this problem, as the code (ASP script) is, more often than not, intermingled with the content (HTML). This makes it difficult for design and coding to be done at the same time, as well as risking potential problems if updates to the page are required.

Code Inline

ASP.NET gets around this problem in one of two ways. The first is the code inline model, where code is still held within the ASP.NET page, but is not mixed with the HTML. It's easy to separate the code and content into two sections. For example:

```
<html>

<%-- This is the code section %>
<script runat="server">

 Public Sub btn_Click(Sender As Object, E As EventArgs)

  YourName.Text = Name.Text
 End Sub

</script>

<body>

<%-- This is the content section %>
 <form runat="server">

  Enter your name: <asp:TextBox id="Name" runat="server"/>
  <br/>

  Press the button: <asp:Button OnClick="btn_Click"
              runat="server" Text="Press Me"/>
  <br/>

  Your name is: <asp:Label id="YourName" runat="server"/>

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

This isn't that radical a design, but it is a difference from ASP where the <%…%> server blocks are often intermingled with the HTML. Don't worry about what the code does for the moment, as we'll be covering that later. What's important is that all of the script is kept separate from the content. This split is possible in ASP.NET because of the new server control architecture, which allows access to the HTML controls from server-based code. We'll be looking at this in a moment.

Code Behind

The second way of separating code from content is the code behind model, where the code is completely removed into a separate file. Using the example we saw above, our HTML file would now look like this:

```
<%@Page Language="VB" Inherits="Ch1CodeBehind"
    Src="Components\Ch1CodeBehind.vb" %>

<html>
<body>

<%-- This is the content section %>
 <form runat="server">

  Enter your name: <asp:TextBox id="Name" runat="server"/>
  <br/>

  Press the button: <asp:Button OnClick="btn_Click"
              runat="server" Text="Press Me"/>
  <br/>

  Your name is: <asp:Label id="YourName" runat="server"/>

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

Once again don't worry too much about the code itself – it's the structure that's important. Notice how the script block has been removed, and a special `Page` directive has been added (these are covered in Chapter 4). This tells the CLR that the current page inherits its code from the named file, which looks like:

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls

Public Class Ch1CodeBehind
       Inherits System.Web.UI.Page

 Public Sub btn_Click(Sender As Object, E As EventArgs)

  YourName.Text = Name.Text

 End Sub

End Class
```

Notice that the procedure `btn_Click` is exactly the same as it was when it was inline. That's one of the great features of the code behind model; apart from a few directives, the code remains exactly the same. And, since we're now working in a compiled environment, there's no performance loss either.

Configuration

Two things govern the configuration of ASP.NET. The first is the standard IIS settings, no different from existing ASP applications. The second is the configuration file, an XML file containing the meta data for our application. There is a machine wide file (`machine.config`) containing the defaults for all ASP.NET applications, and each application can have its own file (`web.config`) to override the defaults. The advantage of a file containing configuration information is that we don't need to touch the registry to modify settings – each application is self-contained. This has an added advantage when we look to deploy an ASP.NET application, because the configuration is just one of the files that we deploy.

The configuration files are covered in detail in Chapter 13.

Deployment

Deployment is another area made significantly simpler in ASP.NET, and is generally called **XCopy Deployment**, for the simple reason that that's all we generally have to do. Each application is self-contained, including the configuration file and components. In the .NET framework, components no longer require registration, and copying them to their target location is all that's required.

Deployment is covered in detail in Chapter 13.

There are exceptions to this model of deployment. One is if we are interacting with COM/COM+ components, which still need to be registered. Another is if we are using Shared Assemblies, where .NET components are being used by more than one ASP.NET application. In this case the component isn't kept within the same directory as the rest of the ASP.NET files.

Interoperability with COM/COM+ is covered in Chapter 23.

Writing ASP.NET Pages

The first part of this chapter has been a brief overview of some of the differences between ASP and ASP.NET, and Chapter 4 goes into this in more detail. Now it's time to show you how to get those ASP.NET pages up and running as quickly as possible. Let's consider a simple form that extracts the author details from the `pubs` database. We'll have a drop down list to show the various states where the authors live, a button to fetch the information, and a grid. This will quickly show you several simple techniques you can use in your pages.

Creating a Web Site

The first thing to do is decide on where you want to create your own samples. Like ASP we can create a directory under `\InetPub\wwwroot`, or create a directory elsewhere and use a Virtual Site or Virtual Directory to point to it. There's no difference between the methods, it's purely a matter of preferences.

Next you can create your web pages, using whatever editor you prefer. You should give them an extension of `.aspx`.

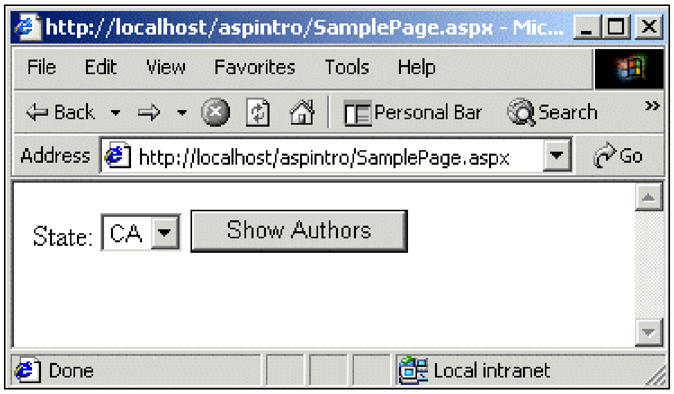
The Sample Page

Now let's add the code for the sample page – call this `SamplePage.aspx` (we'll examine it in more detail after we've seen it running). This page assumes that the Pubs database is installed on your system.

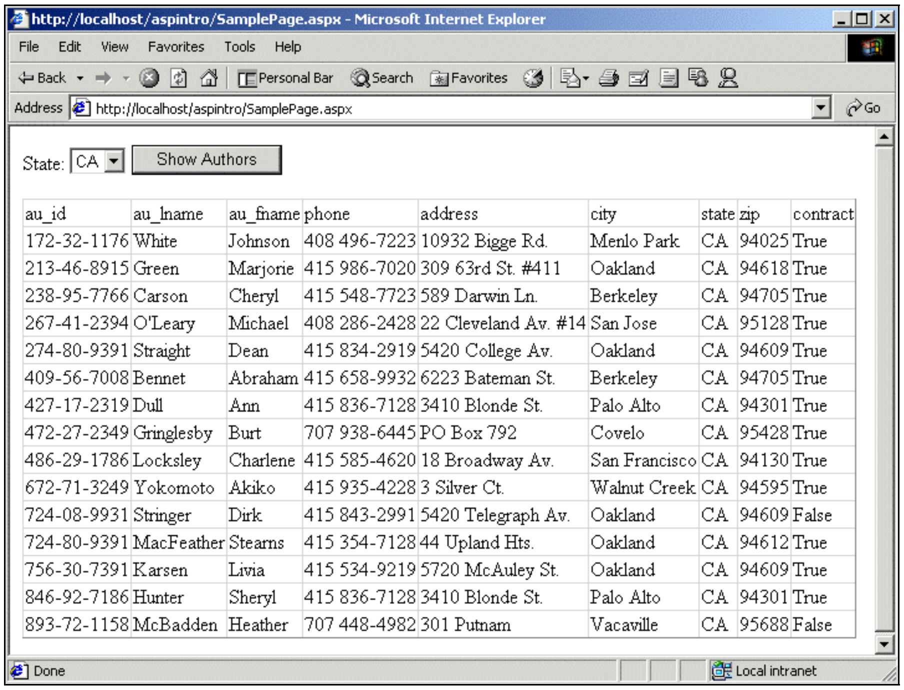
```
<%@ Import Namespace="System.Data.SqlClient" %>

<script language="VB" runat="server">

 Sub Page_Load(Sender As Object, E As EventArgs)

  If Not Page.IsPostBack Then
    state.Items.Add("CA")
    state.Items.Add("IN")
    state.Items.Add("KS")
    state.Items.Add("MD")
    state.Items.Add("MI")
    state.Items.Add("OR")
    state.Items.Add("TN")
    state.Items.Add("UT")
  End If

 End Sub

 Sub ShowAuthors(Sender As Object, E As EventArgs)

  Dim con As New SqlConnection("Data Source=.; " & _
                "Initial Catalog=pubs; UID=sa; PWD=")
  Dim cmd As SqlCommand
  Dim qry As String

  con.Open()
```

```
   qry = "select * from authors where state='" & _
        state.SelectedItem.Text & "'"
    cmd = New SqlCommand(qry, con)

    DataGrid1.DataSource = cmd.ExecuteReader()
    DataGrid1.DataBind()

    con.Close()

  End Sub

</script>

<form runat="server">

  State: <asp:DropDownList id="state" runat="server" />

  <asp:Button Text="Show Authors" OnClick="ShowAuthors" runat="server"/>

  <p/>
  <asp:DataGrid id="DataGrid1" runat="server"/>

</form>
```

When initially run we see the following:


http://localhost/aspintro/SamplePage.aspx - Mic...
File Edit View Favorites Tools Help
Back Personal Bar Search
Address http://localhost/aspintro/SamplePage.aspx Go
State: CA
Show Authors
Done
Local intranet

Nothing particularly challenging here, and when the button is pressed, the grid fills with authors from the selected state:


http://localhost/aspintro/SamplePage.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address http://localhost/aspintro/SamplePage.aspx
Go
State: CA
Show Authors
Done
Local intranet

au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	True
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	True
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	True
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	True
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	True
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	True
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	True
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428	True
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA	94130	True
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	94595	True
724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av.	Oakland	CA	94609	False
724-80-9391	MacFeather	Stearns	415 354-7128	44 Upland Hts.	Oakland	CA	94612	True
756-30-7391	Karsen	Livia	415 534-9219	5720 McAuley St.	Oakland	CA	94609	True
846-92-7186	Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	True
893-72-1158	McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA	95688	False

Again, nothing that couldn't be achieved with ASP, but let's look at the page code, starting with the controls:

```
<form runat="server">

 State: <asp:DropDownList id="state" runat="server" />

 <asp:Button Text="Show Authors" OnClick="ShowAuthors" runat="server"/>

 <p/>
 <asp:DataGrid id="DataGrid1" runat="server"/>

</form>
```

Here we have a form marked with the `runat="server"` attribute. This tells ASP.NET that the form will be posting back data for use in server code. Within the form there is a `DropDownList` (the equivalent of an HTML `SELECT` list) to contain the states, a `Button` (equivalent of an HTML `INPUT type="button")` to postback the data, and a `DataGrid` to display the authors. The button uses the `OnClick` event to identify the name of the server-side code to run when the button is pressed. Don't get confused by thinking this is the client-side, DHTML `onClick` event, because it's not. The control is a server-side control (`runat="server"`) and therefore the event will be acted upon within server-side code.

Now let's look at the remaining code, starting with the `Import` statement. This tells ASP.NET that we are going to use some data access code, in this case code specific to SQL Server.

```
<%@ Import Namespace="System.Data.SqlClient" %>
```

Next comes the actual code, written in Visual Basic.

```
<script language="VB" runat="server">
```

Here is the first real introduction to the event architecture. When a page is loaded, the `Page_Load` event is raised, and any code within the event procedure is run. In our case, we want to fill the `DropDownList` with a list of states, so we just manually add them to the list. In reality, this data would probably come from a database.

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

  If Not Page.IsPostBack Then
    state.Items.Add("CA")
    state.Items.Add("IN")
    state.Items.Add("KS")
    state.Items.Add("MD")
    state.Items.Add("MI")
    state.Items.Add("OR")
    state.Items.Add("TN")
    state.Items.Add("UT")
  End If

 End Sub
```

One thing to note about this code is that it is wrapped in an `If` statement, checking for a Boolean property called `IsPostBack`. One of the great things about the Web Controls is that they retain their contents across page posts, so we don't have to refill them. Since the `Page_Load` event runs every time the page is run, we'd be adding the states to the list that already exists, and the list would keep getting bigger. The `IsPostBack` property allows us to identify whether or not this is the first time the page has been loaded, or if we have done a post back to the server.

Now, when the button is clicked, the associated event procedure is run. This code just builds a SQL statement, fetches the appropriate data, and binds it to the grid.

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

  Dim con As New SqlConnection("Data Source=.; " & _
                "Initial Catalog=pubs; UID=sa; PWD=")
  Dim cmd As SqlCommand
  Dim qry As String

  con.Open()

  qry = "select * from authors where state='" & _
     state.SelectedItem.Text & "'"
```

```
        cmd = New SqlCommand(qry, con)

        DataGrid1.DataSource = cmd.ExecuteReader()
        DataGrid1.DataBind()

        con.Close()

      End Sub

    </script>
```

This code isn't complex, although it may seem confusing at first glance. The rest of the book explains many of these concepts in more detail, but we can easily see some of the benefits. The code is neatly structured, making it easy to write and maintain. Code is broken down into events, and these are only run when invoked. Chapter 4 contains a good discussion of the page events, how they can be used, and the order in which they are raised.

What's also noticeable is that there's less code to write compared to an equivalent ASP page. This means that we can create applications faster – much of the legwork is done by the controls themselves. What's also cool about the control architecture is that we can write our own to perform similar tasks. Because the whole of the .NET platform is object based, we can take an existing control and inherit from it, creating our own, slightly modified control. A simple example of this would be a grid within a scrollable region. The supplied grid allows for paging, but not scrolling.

Summary

This chapter has been a real whistle-stop tour of why ASP.NET has come about, some of the great features it has, and how easily we can create pages. We've looked at:

- The problems of ASP
- Why ASP.NET came about
- The differences between ASP and ASP.NET
- A simple example of an ASP.NET page

ASP is still a great product, and it's really important to focus on why we had to change, and the benefits it will bring in the long term. Initially there will be some pain as you learn and move towards the .NET architecture, but ultimately your applications will be smaller, faster, and easier to write and maintain. That's pretty much what most developers want from life.

Now it's time to learn about the .NET Framework itself, and how all of these great features are provided.