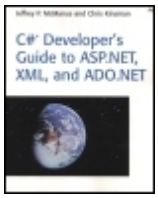
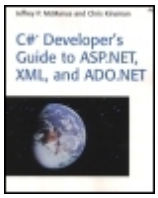


C# Developer's Guide to ASP.NET, XML, and ADO.NET

By Jeffrey P. McManus, Chris Kinsman

Publisher : Addison Wesley
Pub Date : March 29, 2002
ISBN : 0-672-32155-6
Pages : 608
Slots : 1



C# Developer's Guide to ASP.NET, XML, and ADO.NET

By Jeffrey P. McManus, Chris Kinsman

Publisher : Addison Wesley
Pub Date : March 29, 2002
ISBN : 0-672-32155-6
Pages : 608
Slots : 1

The book every Internet application developer working with Microsoft development tools needs to retool their knowledge of the new .NET techniques used to build Windows applications.

- Unbiased, in-depth commentary on the efficacy of the various technologies that comprise .NET as they pertain to Internet database developers.
- Technical know-how without crushing the reader with pointless detail.
- Implementation details that replace and extend the existing Active Server Pages (ASP), XML, and ActiveX Data Object (ADO) functionality currently supported by Microsoft.

Topics covered in this book include: the .NET Foundation Classes that are most used by developers--ASP.NET, XML, and ADO.NET, and details about the construction of Web Services and how they programmatically communicate with each other.

Copyright

05 04 03 02 DOC 4 3 2 1

First printing April 2002

Credits

Associate Publisher

Linda Engelman

Acquisitions Editor

Sondra Scott

Development Editors

Angela Allen

Laurie McGuire

Managing Editor

Charlotte Clapp

Project Editor

Carol Bowers

Copy Editor

Barbara Hacha

Indexer

Tim Tate

Proofreader

Jessica McCarty

Contributor

Anjani Chittajallu

Technical Editor

Joel Mueller

Team Coordinator

Lynne Williams

Media Developer

Dan Scherf

Interior Designer

Gary Adair

Cover Designer

Gary Adair

Page Layout

Ayanna Lacey

Dedication

For Celeste

Jeffrey P. McManus

This book is dedicated to my dad, who supported and encouraged me in everything I did.

Chris Kinsman

About the Authors

Jeffrey P. McManus is a developer and speaker specializing in Microsoft tools. As a developer, he has specialized in online application development using Internet and client/server technologies. He is the author of four books on database and component technologies, including the bestselling *Database Access with Visual Basic 6* (Sams Publishing). Jeffrey regularly speaks at the VBITS/VSLive, European DevWeek, and VBConnections conferences.

Chris Kinsman is a developer and speaker specializing in Microsoft tools. As a developer, he has been responsible for several high-traffic sites based entirely on Microsoft tools, including serving as Vice President of Technology at DevX.com. In addition, Chris spent 10 years consulting with Fortune 500 companies throughout the world to solve their needs by utilizing a variety of Microsoft Visual Studio and Back Office technologies. Chris regularly speaks at the VBITS/VSLive, Web Builder, and SQL2TheMax conferences.

About the Contributor

Anjani Chittajallu obtained a masters degree from Indian Institute of Technology (I.I.T-Madras) with a major in Control Systems Engineering. She specializes in designing and developing enterprise systems with Microsoft Technology. Anjani currently holds MCSD certification. She can be reached at srianjani@hotmail.com.

About the Technical Editor

Joel Mueller is a senior software engineer at DeLani Technologies (www.delani.com), a leading Web development software company, where he has been spearheading the company's Microsoft .NET development effort since July 2000. Prior to the advent of ASP.NET, Joel did extensive work with Microsoft Active Server Pages and Macromedia ColdFusion. He has written for several books and articles on the topics of Macromedia ColdFusion and XML. Joel's current interests include the .NET framework, C#, and sleeping.

Acknowledgments

Jeffrey and Chris would like to extend a special thank you to Anjani Chittajallu, who came through in a pinch and did a bang-up job working on the code examples for this edition of the book. We're grateful to you for your help!

Chapter 1. Introduction: The Need for ASP.NET

IN THIS CHAPTER

Before delving into the particulars of developing with C#, it will be useful to overview ASP.NET. This chapter summarizes ASP.NET's features, including some insight into how they represent improvements over ASP.old.

Problems with ASP Today

When Active Server Pages (ASP) was first introduced almost five years ago, it was seen as an answer to the awkward techniques used at that time for creating dynamic content on the Web. At the time Common Gateway Interface programs or proprietary server plug-ins were the way that most of the Web's dynamic content was created. With the release of ASP 1.0, Microsoft changed all that. ASP 1.0 provided a flexible robust scripting architecture that enabled developers to rapidly create dynamic Web applications. Developers could write in VBScript or JScript and Microsoft provided a number of services to make development easy. At the time, it was just what developers needed. As Web development matured several shortcomings of the platform became evident, and persist until today.

Separation of Code and Design

As the Web grew in popularity in the early 90s, developers experienced three distinct waves of development paradigms. In the first wave, Web developers created static HTML documents and linked them together. This was the era of the "brochure" Web site and was more about looks than anything else. The second wave brought the concept of dynamic content to the fore. Developers started creating registration forms and various small pieces of functionality and adding them into existing Web sites. The third wave was when the first and second waves came together. Web sites were being designed from the ground up to be interactive; they were treated more like an application and less like a magazine with a subscription card in it. In most instances this type of interactive page design created a development paradigm that went like so:

- Designers created page mockups in HTML.
- Developers added code to the pages.
- When designers needed to change their design, they copied and pasted the existing code into the new page, butchering it and destroying its functionality.

The severity of this problem typically depended on the size of the site, the smarts of the designers, and the techniques that developers used to guard against this mangling.

With the release of Visual Studio 6 in September 1998, it was clear that Microsoft recognized this burgeoning problem and attempted to resolve it with a new feature in Visual Basic 6, Web Classes. Web Classes made an attempt to separate the design of a page from the code that interacted with it. It enabled this separation by using an HTML template and providing a facility for doing tag replacement in the template. There were a number of problems with Web Classes. Although a great idea, they suffered from two main issues. First, the Web Classes were implemented entirely in Visual Basic, which required traditional ASP developers to shift their thinking patterns for creating applications. Second, Microsoft had scalability issues related to the threading models of ASP and Visual Basic. Because of the previously stated reasons and many other smaller ones, Web Classes never really gained any traction among developers.

Scripting Language Based

When ASP 1.0 was first released, the fact that all development was done using scripting languages was a big plus. It meant that developers didn't have to go through a painful restart/compile process that they might have been accustomed to with CGI or ISAPI style applications. As applications grew larger, numbers of users increased and developers were using ASP for increasingly difficult problems. The fact that all code was interpreted became a potential performance bottleneck. When using VBScript there was limited support for error handling. Many developers sidestepped this issue by moving code into compiled COM objects. While this move solved some of the performance problems it created new ones in deployment and scalability.

State Management

One of the most frustrating aspects that new Web developers faced early was dealing with the stateless nature of Web development. With ASP 1.0, Microsoft introduced the concept of a Session object, which was designed to make associating state with a particular user easy. This addition was arguably one of the most compelling features of ASP 1.0. Scalability and reliability started to become important as developers began creating larger applications. To address this need, developers started deploying their applications to Web farms. Web farms use multiple servers and spread the request for pages across them somewhat equally. This makes for a great scalability story unless the developer is using that cool Session object. This object is specific to a particular machine in a Web farm and will not work if a user gets bounced to another server. So, an application that was deployed to a Web farm could not use the Session object.

PREVIOUS NEXT

Introducing ASP.NET

ASP.NET is Microsoft's answer to the aforementioned problems and many others that were not explicitly stated. It is a fundamental rewrite of ASP that has been in process for more than two years. The ASP team took a close look at the problems facing Web developers and created a brand new platform in the spirit of traditional ASP to solve those problems. Having used ASP.NET for a considerable amount of time, we can conclusively say they hit a home run with this release.

Platform Architecture

ASP.old was an Internet Server Application Programming Interface (ISAPI) filter that was written specifically to interact with Internet Information Server (IIS). It was monolithic in nature and relied very little on external services.

NOTE

Note: In the IIS 5.0 time frame, ASP did use Microsoft Transaction Server (MTS) as an external service.

ASP.NET is still an ISAPI filter. However, unlike ASP.old, ASP.NET relies on a large number of "external" services—the .NET framework. ASP.NET and the .NET framework are so tightly coupled that it is difficult to consider the .NET framework as an external service. However, since it is accessible from applications outside the scope of ASP.NET, it should be considered an "external" service. As it turns out, this is a huge win for the ASP.NET developer. No longer must the developer write everything from scratch. Instead, the .NET framework provides a large library of prewritten functionality.

The .NET framework redistributable consists of three main parts: the Common Language Runtime, the .NET framework base classes, and ASP.NET.

Common Language Runtime

The Common Language Runtime (CLR) is the execution engine for .NET framework applications. However, despite the common misconception, it is not an interpreter. .NET applications are fully compiled applications that use the CLR to provide a number of services at execution. These services include:

- Code management (loading and execution)
- Application memory isolation
- Verification of type safety

- Conversion of IL to native code
- Access to metadata
- Garbage collection
- Enforcement of code access security
- Exception handling
- Interoperability
- Automation of object layout
- Support for debugging and profiling

The CLR is a platform that abstracts functionality from the operating system. In this sense, code written to target the CLR is "platform independent" provided that there is an implementation of the CLR on the destination platform.

Managed Execution

The CLR isn't just a library or framework of functions that an executing program can call upon. It interacts with running code on a number of levels. The loader provided by the CLR performs validation, security checks, and a number of other tasks each time a piece of code is loaded. Memory allocation and access are also controlled by the CLR. When you hear about "Managed Execution," this is what folks are speaking about: the interaction between the CLR and the executing code to produce reliable applications.

Cross-Language Interoperability

One of the most frustrating things with current COM- or API-based development practices are that interfaces are usually written with a particular language consumer in mind. When writing a component to be consumed by a Visual Basic program, a developer will typically create the interfaces in a different fashion than if the component were intended to be consumed by a C++ program. This means that to reach both audiences, the developer must either use a least common denominator approach to developing the interface or must develop an interface for each consumer. This is clearly not the most productive way to write components. A second problem that most developers merely

accept as normal today is that most components need to be written in a single language. If you create a component in C++ that exposes an employee object, you can't then inherit from that object in Visual Basic to create a Developer object. This means that typically a single language is chosen for most development projects to enable reuse.

.NET changes all this. Cross-language interoperability was built in from the start. All .NET languages must adhere to the Common Language Specification (CLS) that specifies the base level of functionality that each language must implement to play well with others. The CLS is written in such a way that each language can keep its unique flavor but still operate correctly with other languages within the CLR. The CLS includes a number of data types that all conforming languages must support. This restriction works to eliminate a common problem for developers: creating an interface that utilizes data types that another language doesn't support. It also supports both Binary as well as Source code inheritance, enabling the developer to create an Employee object in C# and inherit from it in Visual Basic.

What this means to you as a developer is that code reuse has become much simpler. As long as the code was written for .NET, you don't need to worry what language it was written in. In fact, the choice of language becomes more of a lifestyle choice instead of a capability choice. All languages in .NET are theoretically created equal, so you gain no performance or functionality benefit by using Visual Basic instead of C#. Use the language in which you are the most productive.

New Features in ASP.NET

Up to this point all the features mentioned are gained due to the hosting of ASP.NET on top of the .NET framework. However, these features are just the tip of the iceberg. As I mentioned previously, ASP.NET is a total rewrite, with significant features aside from the intrinsic .NET ones. We are going to give you an overview of the new features in ASP.NET while showing how these features address the problems of separation of code and design, scripting languages, and state management.

Web Forms

Web forms are Microsoft's attempt to solve the problem of the separation of code and design. ASP.NET now offers a code-behind model reminiscent of the forms designer in Visual Basic. This means that you can now place your code in a separate file and still interact with the page. This separation is done by providing a new event-driven model on top of page execution, as well as providing an object model on top of the HTML in the page. Instead of a top-to-bottom linear execution model, events are raised during the execution of a page. Your code sinks those events and responds to them by interacting with the object model that sits on top of the HTML.

This quite neatly solves the issue of designers modifying the HTML and breaking code.

In addition to the new execution model, Microsoft has also created a new server-side control model. Unlike the lame Design Time Controls in Visual Interdev, these new models are incredibly useful encapsulations of common display paradigms. They do not introduce any browser dependencies and they run on the server, not the client. In the few cases where they emit browser-dependent code, they sniff the browser and degrade gracefully. More information on Web forms can be found in Chapter 2, "Page Framework."

Web Services

As we move beyond the first and second generations of Web applications, it has become apparent that the paradigm

of the Web can be extended to solve problems that predate it. In the past, businesses exchanged information using Electronic Data Interchange (EDI) over Value Added Networks (VANs). This worked well enough, but the cost of gaining access to a VAN as well as the complexity of implementing various industry-specific EDI protocols excluded all but the largest companies from participating in the exchange of information.

Web services are a way to transfer the same types of information over the Internet (instead of expensive VANs) using industry-standard protocols such as HTTP, XML, and TCP/IP. Web services are now so easy to create in .NET that individuals or businesses of any size should be able to play in this space. Web services aren't limited to replacing traditional EDI protocols. They open up many opportunities that EDI has never made inroads into. Jump ahead to Chapter 6, "Web Services," for more information.

Data Access

When ASP 1.0 first shipped, the data access story at Microsoft was in a state of flux. At the time, Remote Data Objects (RDO) was the technology of choice for Visual Basic developers. ActiveX Data Objects (ADO) was introduced with the shipment of Visual Basic 5.0 in February 1997. It was intended to be a new data access model for all types of data and was paired with another new technology, OLE DB.

While ADO was great for what it was designed for—connected data access—it fell short in the disconnected arena. Features were added in successive versions to allow it to work in a disconnected fashion. ADO 1.0 had no support for XML. ADO 1.0 could not predict the success of XML as a data description language when it was shipped, and XML support was cobbled onto later versions. Neither of these features were designed in from the beginning.

ADO.NET is a new data access technology taking advantage of all the things Microsoft learned with ADO, RDO, OLEDB, ODBC, and other preceding data access implementations. It was designed from the beginning to be coupled very tightly to XML and work effectively in a disconnected fashion. For more information see Chapter 11, "Creating Database Applications with ADO.NET."

Deployment

One of the perennial arguments among ASP developers was how much code to migrate into COM objects. Some writers advocated all code living in COM objects and ASP should only contain a single-method call to invoke the COM object. While this might have been great in theory, it eliminated one of the biggest strengths of ASP: the capability to rapidly create an application and make changes on-the-fly. With all the logic and HTML tied up in COM objects, a simple HTML tag change meant recompiling and redeploying the COM objects. The biggest problem in our minds lies with using this approach. COM objects are Dynamic Link Libraries (DLL) that are dynamically loaded by IIS. While loaded they cannot be replaced. To deploy a COM object the developer needed to shut down IIS, shut down the MTS packages the COM object lived in, replace it, and then restart IIS. This summary is actually a simplification of the process, but you can see the problems this technique brings to the fore. Each time a new version is deployed, the Web server must go down! The downtime this technique causes can be handled by creating Web farms and doing rolling upgrades; however, in a large Web farm this means a simple change can take literally hours to deploy as the new objects are rolled out.

With the code-behind model inherent in ASP.NET, this situation could have been exacerbated. Instead, Microsoft vastly simplified the deployment model. Comp-onents, now called assemblies, no longer require registration on a machine for deployment. Assemblies are the .NET equivalent of a COM object. They are self describing and contain a manifest which contains metadata about the assembly. The metadata includes things like the version, the assemblies it depends on, and potentially, its security identity.

Deployment is as easy as copying the assemblies into a /bin directory in the application root. ASP.NET will notice that a new version has been copied over and unload the old version and load the new version! Deployment becomes as simple as an XCOPY /S or a recursive FTP to upload the new files to the Web server. For more information see Chapter 5, "Configuration and Deployment."

Configuration

In the past, all configuration information for ASP was stored as part of the IIS metabase. This was a binary file analogous to the registry that held all settings for IIS and ASP. The only ways to effect changes were to use the Internet Services Manager or to write scripts that utilized the Active Directory Services Interfaces (ADSI) to automate the changes. This process made it very difficult to synchronize the settings of multiple servers in a Web farm.

ASP.NET introduces a new paradigm for all settings. Instead of being stored in the opaque metabase, they are now stored as a hierarchical set of XML configuration files. These files live in the application root and subdirectories. So, now as a developer uses XCOPY to deploy source files, the settings are also deployed! No need to write a bunch of configuration scripts anymore. For more information see Chapter 5.

State Management

State management has been vastly improved in ASP.NET. Now, three options exist for maintaining state on the server. The classic ASP 3.0 method of in-memory state on a single server still exists. In addition, an out-of-process state server and a durable state option is stored in SQL Server.

The other limitation of state services in ASP.old was the reliance on cookies for connecting a user back up to their state. ASP.NET introduces a new option for cookieless state that performs URL munging to connect a user to state information. For more information see Chapter 4, "State Management and Caching."

Caching

The reason most developers use ASP is to lend a dynamic nature to the Web. This could mean accessing backend databases for data or perhaps pulling it in from nontraditional backends. The problem with this dynamic content is that while one can easily scale the Web tier using a scale-out methodology of multiple Web servers, this scaling is not as easily done in the data tier. Scale-out approaches for databases are just beginning to appear. Until these approaches are perfected, how can Web developers scale applications?

For data that changes infrequently, caching is a great solution. ASP.NET offers two forms of caching. Output caching takes an entire page and stores the executed results in memory for later delivery. Data caching takes items that were expensive to create, such as DataSets, and caches them on the server side. For more information see Chapter 4.

Debugging

Debugging ASP applications has always been difficult. While remote debugging solutions were offered in previous versions of Visual Studio, precious few developers were able to get them to work consistently. Consequently, most debugging consisted of Response.Write statements littered throughout code or some type of logging mechanism that the developer created.

ASP.NET not only improves remote debugging and makes it consistent, it also offers a new Trace facility that is great for handling the types of things that logging or Response.Write were used for in the past. For more information see Chapter 3, "Debugging ASP.NET Applications."

Availability

Anybody that currently has a site on the Web knows that availability is key. If your site is down, a visitor can turn to a million others. Problem is, they might not come back!

ASP.NET has introduced a number of process controls that are aimed directly at improving availability. Now the ASP.NET process can restart automatically based on things such as memory utilization, time up, or even number of requests handled, which helps cope with situations where ASP used to get jammed up. For more information see Chapter 5.