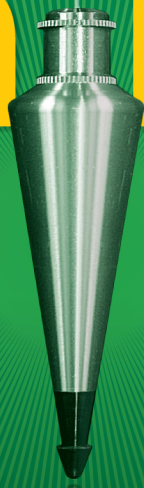
Microsoft

Designing and Developing Web Applications Using Microsoft® .NET Framework 4

Tony Northrup

MCPD

Exam Ref



EXAM

70-519

Exam Ref 70-519

Designing and Developing Web Applications Using Microsoft® .NET Framework 4

Professional-level prep for the professional-level exam.

Prepare for MCPD Exam 70-519—and help demonstrate your real-world mastery of web application design and development with .NET Framework 4. Designed for experienced, MCTS-certified professionals ready to advance their status—*Exam Ref* focuses on the critical-thinking and decision-making acumen needed for success at the MCPD level.

Focus on the expertise measured by these objectives:

- Designing the Application Architecture
- Designing the User Experience
- Designing Data Strategies and Structures
- Designing Security Architecture and Implementation
- Preparing for and Investigating Application Issues
- Designing a Deployment Strategy

Exam Ref features:

- Focus on job-role expertise
- Organized by exam objectives
- Strategic, what-if scenarios
- 15% exam discount from Microsoft. Offer expires 12/31/2016. Details inside.

CERTIFICATION

The *Microsoft Certified Professional Developer* (MCPD) certification helps validate the comprehensive skills needed to develop applications using Microsoft Visual Studio®, the .NET Framework, and other development technologies.

JOB ROLE

Professionals certified as *MCPD Web Developer 4* build interactive, data-driven ASP.NET applications for both intranets and the Internet.

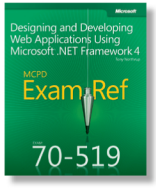
REQUIRED EXPERIENCE

Successful candidates generally have three or more years of real-world experience.

See full details at:
microsoft.com/learning/certification

MEET THE FAMILY

- Train
- Prep
- Practice



- Prep
- Optional Practice*

** Select titles coming soon*

- Review

About the Author

Tony Northrup, MVP, MCPD, MCITP, MCSE, CISSP, is a consultant and the author of more than 25 books on Windows and web development, networking, and security.

ISBN: 978-0-7356-5726-7

U.S.A. $39.99
Canada $41.99
[*Recommended*]

Certification/
Microsoft Visual Studio

Microsoft® Visual Studio® 2010

Microsoft®

Microsoft®

MCPD 70-519 Exam Ref:

Designing and Developing Web Applications Using Microsoft® .NET Framework 4

Tony Northrup

ISBN: 978-0-7356-5726-7

2 3 4 5 6 7 8 9 10 QG 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at *mspinput@microsoft.com*. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Acquisitions and Developmental Editor: Ken Jones

Production Editor: Adam Zaremba

Editorial Production: Octal Publishing, Inc.

Technical Reviewer: Bill Chapman

Copyeditor: Roger LeBlanc

Indexer: Denise Getz

Cover Composition: Karen Montgomery

Illustrator: Robert Romano

[2013-01-25]

For my favorite nephews and niece: Tyler, Austin, and Mya Rheaume

Contents at a Glance

Contents

Introduction

Most development books take a very low-level approach, teaching you how to use individual classes and accomplish fine-grained tasks. Like the Microsoft 70-519 certification exam, this book takes a high-level approach, building on your lower-level web development knowledge and extending it into application design. Both the exam and the book are so high-level that there is very little coding involved. In fact, most of the code samples this book provides simply illustrate higher-level concepts.

The 70-519 certification exam tests your knowledge of designing and developing web applications. By passing the exam, you will prove that you have the knowledge and experience to design complex web applications using Microsoft technologies. This book will review every concept described in the exam objective domains:

- Design application architectures
- Design the user experience
- Design data strategies and structures
- Design a security architecture and implementation
- Prepare for and investigate application issues
- Design a deployment strategy

This book covers every exam objective, but it does not necessarily cover every exam question. Microsoft regularly adds new questions to the exam, making it impossible for this (or any) book to provide every answer. Instead, this book is designed to supplement your relevant independent study and real-world experience. If you encounter a topic in this book that you do not feel completely comfortable with, you should spend several hours researching the topic further using MSDN, blogs, and support forums. Ideally, you should also create a practical application with the technology to gain hands-on experience.

Microsoft Certified Professional Program

Microsoft certifications provide the best method for proving your command of current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies. Computer professionals who become Microsoft certified are recognized as experts and are sought after industry-wide. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **OTHER MICROSOFT CERTIFICATIONS**

For a full list of Microsoft certifications, go to *www.microsoft.com/learning/mcp/default.asp*.

Acknowledgments

First and foremost, I'd like to thank Ken Jones at O'Reilly for his work in designing the Microsoft Press Exam Ref book series, for choosing me (once again) as an author, and for his work as an editor. It's been great to work with you, as always, Ken!

I'd also like to thank Bill Chapman, the Technical Editor, Adam Zaremba, the Production Editor, Dan Fauxsmith, the Production Manager, and Roger LeBlanc, the Copy Editor.

Finally, I must thank my friends and family for their support, especially Eddie and Christine Mercado (for letting me use of their home after hurricane Irene), Brian and Melissa Rheaume (for taking me to Greenport on their boat), Jose and Kristin Gonzales (for the many laughs), Chelsea and Madelyn Knowles (for their patience while I worked too much during the Summer), and Papa Jose and Nana Lucy (for the meat pies).

Support & Feedback

The following sections provide information on errata, book support, feedback, and contact information.

Errata

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://www.microsoftpressstore.com/title/ 9780735657267

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in Touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*

Preparing for the Exam

Microsoft certification exams are a great way to build your resume and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. Although there is no substitute for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. We recommend that you augment your exam preparation plan by using a combination of available study materials and courses. For example, you might use the Exam Ref and another study guide for your "at home" preparation, and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you.

CHAPTER 1

Designing the Application Architecture

The highest level aspect of the design process is also the most exciting: designing the application architecture. In this stage, the application begins to come to life, and you are not getting bogged down in technical details. You create a logical design for your application and then map the logical layers to physical servers. After you determine the physical layout, you can choose interapplication communication mechanisms and plan for cross-cutting concerns, such as systems administration.

IMPORTANT

Have you read page xix?

It contains valuable information regarding the skills you need to pass the exam.

Further into the design process, you choose how presentation logic will be divided between the client and server. For client-side components, you will need to decide between basic JavaScript, jQuery, Microsoft AJAX, and Microsoft Silverlight. For server-side components, you will need to choose between HTML controls, server controls, user controls, and Web Parts.

Finally, you will need to decide how to implement various state-management tasks. The Microsoft .NET Framework provides a wide variety of technologies, including application state, session state, view state, cookies, and caching.

Objectives in this chapter:

- Objective 1.1: Plan the division of application logic
- Objective 1.2: Analyze requirements and recommend a system topology
- Objective 1.3: Choose appropriate client-side technologies
- Objective 1.4: Choose appropriate server-side technologies
- Objective 1.5: Design state management

Real World

The application design process starts when management determines that a new application can fulfill a business requirement. As management describes what they need from the new application, your mind will race with all the reasons the application won't work the way they want. Pointing out every potential problem might feel like you're demonstrating your technical skill and preventing future frustrations, but in the real world, it hinders the design process, dampens creativity, and annoys management.

As developers, our minds have been tuned to spot and eliminate flaws. However, you need to be creative and positive during the application design process. Do your best to ignore the low-level challenges; troubleshooting is a job for coders. Designers must create.

Objective 1.1: Plan the Division of Application Logic

In the early days of the web, browsers did little more than render HTML and display images. Today, thanks to technologies such as JavaScript, Flash, and Silverlight, the browser can interact with the user, validate data, and communicate with servers without loading new webpages. Use these client-side capabilities properly, and you can make your web application feel faster, reduce bandwidth, and reduce user input errors.

Server-side processing still has its place, however. First, server-side code is much easier to develop, test, and maintain. Second, anything but the most trivial data validation must be performed on the server, because it is possible for malicious attackers to bypass client-side validation. Third, some clients do not support JavaScript, Flash, or Silverlight, requiring you to duplicate any mandatory client-side functionality on the server.

This objective covers how to:

- Choose whether to implement functionality on the client or server.
- Efficiently use client-side scripting languages.
- Explain the capabilities and drawbacks of rich, client-side plug-ins such as Flash and Silverlight.
- Partition applications according to the separation of concerns principle.
- Plan for long-running processes.

Choosing Between the Client Side and Server Side

Many tasks can be performed at either the client or the server. For example, if you ask the user to enter his address in a web form, you can provide a *DropDownList* named *Country DropDownList* that contains every country/region in the world. When the user selects a country, you can populate the *StateDropDownList* with a list of states or provinces in his country.

You can do this on either the server or the client:

- **Server** In ASP.NET, set *CountryDropDownList.AutoPostBack* to *True*. In the *DropDownList.SelectedIndexChanged* event handler, populate *StateDropDownList*.
- **Client** Create a JavaScript function that handles the *CountryDropDownList.OnChange* JavaScript event and populates the *StateDropDownList* on the client.

Neither approach is clearly superior, but they each have advantages. By populating the list on the server side, you keep more code in ASP.NET, which is generally easier to write, troubleshoot, and maintain than JavaScript. Additionally, server-side processing works when the client does not support JavaScript.

By populating the list on the client side, you improve performance for both the user and the server. Client-side processing avoids a browser postback to the server when the user selects her country. This eliminates a delay in data entry that could last several seconds. Additionally, by reducing the number of requests sent to the web server, it reduces the performance impact on the server, thus improving scalability.

EXAM TIP

The 70-519 exam does not require you to know JavaScript or Microsoft AJAX; those topics were covered by the 70-515 exam. In fact, the 70-519 exam does not require you to know how to write code at all. You do need to know the capabilities and limitations of JavaScript and AJAX, however, and have a higher-level understanding of the impact of writing different types of code.

Table 1-1 compares common tasks that can be performed at either the client or server, and how you write code to accomplish them. When validating user input, you typically validate it on the client (for immediate responsiveness) and again at the server (for security and for browsers that do not support JavaScript).

TABLE 1-1 Performing Different Tasks at the Client-side and Server-side

Task	Client-side feature	Server-side feature
Respond to a button click	JavaScript's *onClick* event	ASP.NET's *Button.Click* event
Access a SOAP web service	JavaScript SOAP clients or the *XMLHttpRequest* object	Import the definition, and access the methods directly
Update part of a page with data from the server	ASP.NET *UpdatePanel* control	Any server control

Task	Client-side feature	Server-side feature
Validate user input	*RequiredFieldValidator*, *RangeValidator*, *RegularExpressionValidator*, and *CustomValidator* (with the *ClientValidationFunction* property)	*RequiredFieldValidator*, *RangeValidator*, *RegularExpressionValidator*, and *CustomValidator* (with the *OnServerValidate* property)

Many tasks should always be done on the server, while other tasks should be performed on the client (when the client supports JavaScript). Table 1-2 lists tasks that can be done on the client, and situations that require you to perform the task on the server, instead.

TABLE 1-2 Client-side and Server-side Tasks

Client-side Tasks	Server-side Tasks
For convenience, notify users if they enter data in an invalid format. For example, if they enter too few numbers for a credit card.	For security and data integrity, verify that user data falls within specified bounds.
Dynamically add items to a menu, based on what the user does within a single webpage.	Add items to a menu for restricted pages that only authorized users can access.
Perform tasks that require access to the client computer, such as saving files using JavaScript or accessing the graphical processing unit (GPU) with Silverlight.	Perform tasks that require access to resources on the internal network that the server can access but are not exposed to the client.
Perform tasks that consume a great deal of bandwidth when communicating between the client and server.	Perform tasks that cannot be performed on the client, especially when the client lacks JavaScript, Flash, or Silverlight.
Process business logic that the end user is allowed to examine (because the user can access the source code).	Process business logic that should not be exposed to the end user.
Perform user-interface interactions, such as expanding menus and displaying slide shows.	Perform security-oriented tasks, such as processing credit cards and authenticating users.

If a task can be performed on either the client or the server, you should perform the task on the server because server-side programming is more efficient, the code is easier to debug, and the application is easier to maintain. Table 1-3 describes the key differences between client-side and server-side programming.

TABLE 1-3 Comparison of Client-side and Server-side Programming

Client-side Programming	Server-side Programming
Code is written in Microsoft Visual Studio 2010 with limited support for auto-complete.	Code is written in Visual Studio 2010 with full support for auto-complete, descriptions of all parameters, and integrated documentation.
Weak typing and run-time detection of errors.	Strong typing with compile-time detection of many errors

Client-side Programming	Server-side Programming
Must test in every supported operating system, browser, and browser version (which can be more than a dozen different environments).	Only need to test in a single-web-server environment.
Somewhat imprecise debugging provided by Microsoft Internet Explorer and Visual Studio 2010. Other browsers require browser-specific debugging tools.	Precise debugging provided by Microsoft Internet Information Services (IIS) and the Visual Studio 2010 ASP.NET runtime environment.
Code might never run if the client does not support JavaScript.	Code always runs regardless of client capabilities.
End users can view, manipulate, or bypass code.	Code is never exposed to the end user.

Partitioning According to Separation of Concerns

Separation of Concerns (SoC) is a software architecture concept for dividing code used for different purposes. For example, if you were designing a web application with SoC in mind, you might create different application layers for the user interface, the business logic, the data access, and the database itself.

Microsoft's early web development languages provided little opportunity for implementing SoC. However, the importance of SoC is reflected in each new web development model that Microsoft has released, as the following sections describe.

Classic ASP

In 1998, Microsoft released Active Server Pages (ASP), now known as Classic ASP. Classic ASP mixed the HTML user interface and all back-end code into a single file. To write output to part of a webpage, you had to write code at the appropriate spot in the HTML:

```
<p>First name:
<%
    Dim firstName
    firstName = "Kim" (Akers)
    Response.Write firstName
%>
</p>
<p>Last name:
<%
    Dim lastName
    lastName = "Akers"
    Response.Write lastName
%>
</p>
```

Because all the code was mixed together, a web designer who wanted to modify the user interface might accidentally change code that performed business logic or accessed the database. Similarly, if a database designer changed the layout of a table in the database, it might

affect the user interface of the application. Performing quality assurance (QA) was difficult because you could not easily test individual components. Instead, developers had to simulate user input and then examine the resulting HTML output for an expected result. Different developers could not easily work on the same page at the same time.

ASP.NET

In 2002, Microsoft released ASP.NET, which allowed developers to use code-behind files to separate the HTML and the placement of server controls from the back-end code. This was a definite improvement for implementing SoC, but developers still created a single class for displaying output and responding to user input. This approach makes testing difficult because testing an individual page requires creating an instance of the page class, its child controls, and all dependent classes.

ASP.NET MVC

In 2009, Microsoft released ASP.NET MVC, which is named for the Model-View-Controller software architecture and provides three different layers of SoC:

- **Model** The data and behavior of the application
- **View** The user interface, which displays data provided by the model
- **Controller** Accepts user input, and calls the model and view to generate a response

Figure 1-1 shows the MVC design pattern and the communications between the layers.

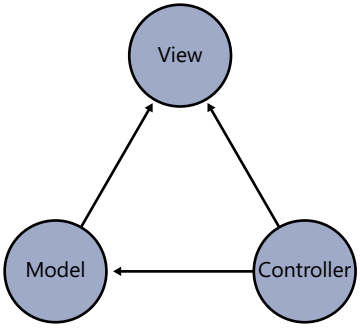


FIGURE 1-1 The MVC design pattern

By providing SoC, MVC provides several benefits. Support for test-driven development allows QA personnel to query the model directly to verify that it provides an expected output when given a specific input. Developers can modify views to update the user interface without any potential impact on the business logic or data access layers. Controllers completely abstract requests from the models and views responding to the request, allowing web architects to specify a structure for the user interface without defining the application architecture.

Implementing SoC can increase development time for smaller applications, albeit by a small margin. However, SoC can dramatically reduce debugging, QA, and maintenance time.

SoC also simplifies dividing development tasks between multiple developers. Therefore, the larger the development effort, the more important SoC becomes.

Planning for Long-Running Processes

Web users are impatient and will cancel a request or give up on a website entirely if pages do not load quickly. As a result, webpages typically need to be rendered in less than a second. That is enough time to query a database or a web service, but performing a longer-running task requires multiple requests.

Consider a travel agency web application that provides flight information from multiple airlines. If a user requests information about all flights between Boston and Chicago on a specific day, the web application might need to send web service requests to a dozen different airlines and wait for the responses before displaying the results to the user. One airline might respond in half a second, but another airline might take 10 seconds to respond.

If the web application queried each airline synchronously (in sequence, one after another), the response time would be delayed by the sum total of all the airline web services. It is more efficient to submit the web service queries asynchronously (in parallel, all at the same time). Then the response time is delayed only by the time required by the slowest web service.

When you create a method, such as a *Button.Click* event handler, the code in the method runs synchronously by default. In other words, the common language runtime (CLR) runs one line of code, waits for the results, and then moves on to the next line. This linear flow is easy for developers to understand, and it is efficient for short-running processes.

If you have long-running processes, such as waiting for a web service to respond, you can use asynchronous processing to allow the .NET Framework to perform other tasks instead of waiting for the response. When the asynchronous response is complete, you can retrieve the results and update the response to the user.

> ***NOTE*** **WRITING ASYNCHRONOUS CODE**
>
> **Ideally, any task that is not dependent on the results of other tasks should be performed asynchronously. Using asynchronous programming techniques improves performance and scalability. In practice, however, you need to weigh the benefits against the complexity of writing and maintaining asynchronous tasks.**

Designing a Webpage for a Long-Running Process

Figure 1-2 shows the typical flow of a synchronous webpage. With this model, however, the user gets no feedback until the server finishes rendering the response. If it takes the server more than a few seconds, the user is likely to cancel the request.

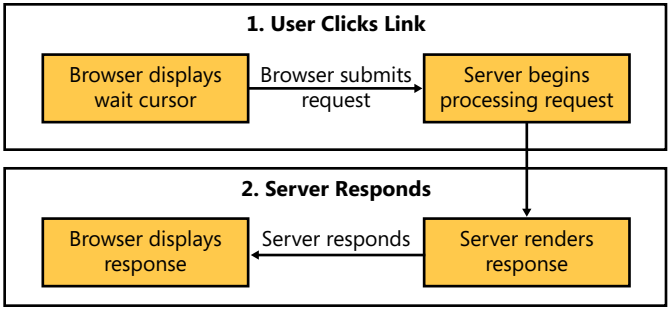


FIGURE 1-2 The flow of a typical synchronous webpage

Figure 1-3 shows the typical flow of an asynchronous webpage. With this model, the server informs the user that the response will take a few moments. A client-side script regularly checks the server to determine whether the results are ready. When the long-running, asynchronous process has completed, the final results are displayed to the user.

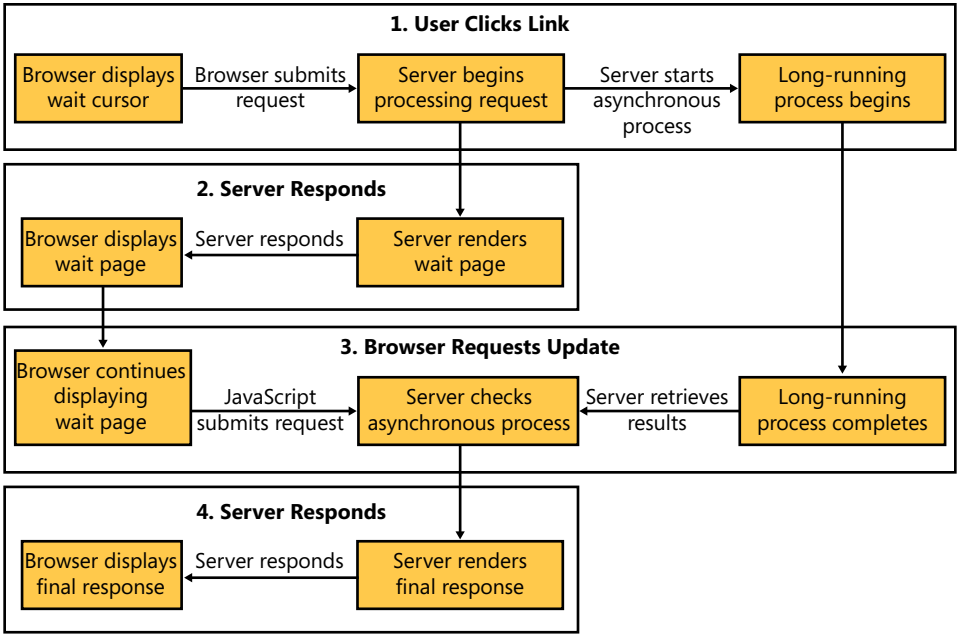


FIGURE 1-3 The flow of a typical asynchronous webpage

You can run a long-running process while remaining responsive to users. In the travel agency example, developers might take one of these two approaches:

- Display a loading page with a progress bar or other animation that shows the application is currently processing the request. This page uses JavaScript to communicate with the server. When the server reports that the results are ready, JavaScript loads the results page.

- Immediately display a formatted results page. Instead of showing the results, a progress bar indicates that the results are loading. In the background, the page runs JavaScript to connect to the server and wait for results. As the server returns results (either partially or all at once), JavaScript adds the results to the page.

For processes that might take more than a minute or two to complete, gather the user's email address and send her a notification with a link to retrieve the results.

Designing a Web Service for a Long-Running Process

Whereas web applications must render the HTML that the browser displays as the user interface, web services return raw data that the client application processes. Because the web service client creates the user interface, the web service developer does not need to decide how to communicate the delay to the user.

The web service developer does, however, need to design the web service to accommodate long-running asynchronous processes. If both the client and server are based on the .NET Framework, and the client is not protected by a firewall or Network Address Translation (NAT) device, you can use *WSDualHttpBinding*, *netTcpBinding*, *NetNamedPipeBinding*, *NetPeerTcpBinding*, or *NetTcpContextBinding* to create a callback contract on the client and then use that callback to notify the client that the process is complete.

> ***NOTE*** **DUPLEX HTTP FOR SILVERLIGHT AND .NET CLIENTS**
>
> **Silverlight clients can use *PollingDuplexHttpBinding*, which supports duplex communications and allows the client to be located behind a firewall or NAT device. Unfortunately, .NET 4.0 does not include a polling duplex HTTP binding. However, you can download a sample custom channel that might suit your needs at *http://archive.msdn.microsoft.com/duplexhttp*.**

If the binding type does not support duplex communications, or you must communicate through a firewall that prevents incoming connections to the client, you should handle long-running web service requests by immediately providing a token the client can use to later retrieve the results. To provide better feedback to the end user, you can also provide an estimated wait time that the client can use to display the progress to the user. Then have the client regularly poll the server to determine if the process is complete.

Use polling to retrieve the results of a long-running query that is using a web service by following this process:

1. The client sends the request to the web service. This might be, for example, "List all flights between Boston and Chicago on May 1."
2. The web service provides a unique token to the client and an approximate wait time. The token should be large and cryptographically random, such as a 20-byte value generated by *RngCryptoServiceProvider*. The wait time should be based on the actual wait time for similar requests.

3. The web service client displays a progress bar to the user to let him know the request is continuing and the application is responsive. The web service asynchronously calls a method to process the request and store the results in a database record associated with the token.

4. After an interval (for example, one-quarter of the estimated wait time), the web service client queries the web service for the results, providing the unique token. If the results are ready, the web service client formats and displays the data; otherwise, it repeats this step.

Objective Summary

- Use client-side scripting to provide users with a rich, responsive interface. However, you must write server-side code when security is important. Additionally, server-side code is more efficient to write, test, troubleshoot, and maintain.
- SoC simplifies development, testing, and updating of large-scale web applications. Strive to design applications with SoC dividing the functional layers of an application.
- To perform a long-running request while appearing responsive to the user, divide a request into multiple steps. In the first step, launch the long-running process asynchronously and display a wait page to the user. Embed JavaScript in the wait page that queries the server for the status of the long-running process and retrieves the final results page when processing is complete.

Objective Review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are designing an ASP.NET web application that allows members of the Administrators role to edit content by using a HyperLink control on each page named *EditHyperLink*. You do not want nonadministrators to discover the location of the administration pages. Which approach should you recommend?

   A. In the ASPX page, set the *EditHyperLink.Visible* property to *True*. In the JavaScript *window.onload* event handler, set the link's *style.display* property to none if the user is not a member of the Administrators role.

   B. In the ASPX page, set the *EditHyperLink.Visible* property to *False*. In the JavaScript *window.onload* event handler, set the link's *style.display* property to block if the user is a member of the Administrators role.

C. In the ASPX page, set the *EditHyperLink.Visible* property to *False*. In the *Page.Load* event handler, set the *HyperLink.Visible* property to *True* if the user is a member of the Administrators role.

D. In the ASPX page, set the *EditHyperLink.Visible* property to *False*. In the *EditHyperLink.Click* event handler, set the *HyperLink.Visible* property to *False* if the user is not a member of the Administrators role.

2. You are designing an ASP.NET web application that provisions virtual machines for a testing environment. Users can provision from 1 to 10 virtual machines at a time, and each virtual machine provision might take up to 60 seconds. For security reasons, the server hosting the virtual machines allows provisioning requests only from the web server. You need to design the application to keep users notified of the provisioning progress. Which approach should you recommend?

   A. On the server, start asynchronous processes to provision each virtual machine. On the client, use JavaScript to query the server every five seconds for a status update.

   B. On the server, synchronously provision each virtual machine. When complete, return a status update to the user.

   C. On the server, calculate the approximate total provisioning time. On the client, use JavaScript to connect to the server hosting the virtual machines and initiate the provisioning.

   D. On the client, use JavaScript to launch a separate asynchronous process for each virtual machine to be provisioned. Within each process, request a page from the web server that provisions a virtual machine.

3. You are creating a new website for an enterprise organization. The enterprise has a quality assurance team that requires developers to use test-driven development. Additionally, the application architecture must partition according to the principle of SoC. Which template should you use?

   A. Use the ASP.NET 4.0 web application project template.

   B. Use the ASP.NET MVC 2 web application project template.

   C. Use the Silverlight application project template.

   D. Create an ASP.NET 4.0 website.

THOUGHT EXPERIMENT
Moving a Site from an Intranet to the Internet

In the following thought experiment, you apply what you've learned about the "Plan the Division of Application Logic" objective to predict how a theoretical website architecture will perform. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a developer for City Power & Light. You are working with management to assess the impact of moving an intranet application to the Internet. The application was created using ASP.NET 2.0. Only authorized and authenticated employees located on the high-speed intranet are allowed to use the application. Employees can enter a customer's identification number or street address, and then examine that customer's power usage over time by viewing a list of monthly statistics or a graphical chart. Typically, employees interpret the information over the phone when a customer calls and requests information about his bill.

Because many of the employee computers have low-powered processors and outdated browsers, the application was designed without any JavaScript or client-side logic. Employees complain that during peak hours, it can take five or ten seconds to load a page with a chart. Showing charts for different time periods requires waiting for a new page to load.

Management needs to give customers direct access to their usage information. Answer the following questions about the future performance of the application:

1. Which factors currently limit the responsiveness of the site: client processing, server processing, client bandwidth, or server bandwidth? How could you improve the performance?
2. How will the website perform if the company provides customers access to it across the Internet without modifying the application?
3. How would you create a reasonably accurate estimate of the server processing capabilities and the amount of bandwidth the site might need on the Internet?
4. How would you reduce the amount of server processing time required to generate the charts?
5. How could you avoid reloading the entire webpage when changing the time period of a chart?
6. How would generating charts on the client affect the site's performance? Which client-side technology would you use? How would you provide the raw data to the client?
7. How would using a content delivery network (CDN) reduce Internet bandwidth requirements? How might a CDN speed delivery of server-side or client-side charts?

Objective 1.2: Analyze Requirements and Recommend a System Topology

Large-scale Internet and intranet applications require you to create a logical design, map it to a physical server architecture, and choose how the different layers will interact. This objective includes an overview of common system topologies, describes how to select a binding for interapplication interactions, helps you choose a binding type, and provides best practices for cross-cutting concerns.

This objective covers how to:

- Design a system topology.
- Design interactions between applications.
- Map the logical design to the physical implementation.
- Validate nonfunctional requirements and cross-cutting concerns.
- Evaluate baseline needs.

Designing a System Topology

There are two system topologies with which you should be familiar: MVC (as described in Objective 1.1), and the three-tier architecture. The three-tier architecture consists of the following:

- **Presentation** The user interface. This tier is responsible for layout and formatting.
- **Application Logic** Also known as Business Logic, this tier is responsible for making decisions based on business rules. If this tier has multiple layers, you can refer to the architecture as an *n-tier* architecture.
- **Data** Typically implemented by a database, this tier is responsible for storing and retrieving information.

For a typical ASP.NET web application, the three-tier architecture might be implemented as follows:

- **Presentation** An IIS web server with an ASP.NET web application. The web application retrieves data from the logic tier by using Windows Communication Foundation (WCF), performs minor formatting and processing, adds it to a webpage, and returns it to the client's web browser.
- **Application Logic** A .NET Framework application exposing interfaces via WCF. The logic tier receives requests from the presentation tier, such as "How long will it take to ship this item?" or "Should I offer this customer a coupon?" The logic tier retrieves all data required to answer queries from the data tier.

- **Data** A database server, such as Microsoft SQL Server. The data tier stores raw data, such as a table containing every item for sale and the number of items in inventory or a table with every customer and an index of their orders.

Whether you implement an MVC architecture, a three-tier architecture, or an n-tier architecture, you benefit from these advantages:

- **Easier to divide among different developers** Let the database guys write the logic tier and the design guys write the presentation tier.
- **Easier to replace a single component** For example, you could use existing Linux web servers for the initial deployment and later migrate to Windows. By separating the presentation layer, you would not have to rewrite the entire application—just the presentation.
- **Easier to scale** You can add more web servers without making any changes to the presentation layer.
- **More flexibility** Most web applications include logic and presentation in a single assembly. By separating the two, you simplify replacing the user interface. It also allows you to support multiple, different user interfaces, such as web, Microsoft Windows, and mobile interfaces.
- **Easier to test** The only way to create a reliable application is to create a testable application. As described in Objective 1.1, providing SoC allows you to more easily test individual components.

If these three tiers aren't familiar to you as a web developer, it is because most web applications use a two-tier logical architecture that combines presentation and logic into a single set of classes. Unless you go out of your way, ASP.NET applications (other than MVC applications) use a two-tier architecture.

Designing Interactions Between Applications

Modern web applications are rarely isolated to themselves. They query databases, web services, and other applications to retrieve the data they need. They also provide updates by initiating order processing and signaling support.

Whether the communications are between different applications or different tiers within a single application, you should use WCF to implement it. WCF provides a powerful, flexible, and (optionally) standards-based way to communicate between processes, either on the same computer or across the network.

WCF supports many types of network protocols, implemented as bindings. If you need to be able to communicate across the Internet, choose one of the following HTTP bindings because HTTP communications are almost always allowed through firewalls:

- **wsHttpBinding** A standards, SOAP-based web service, *wsHttpBinding* is perfect when you will be communicating with .NET Framework–based hosts or if you need to communicate across the Internet, where firewalls might block non-HTTP traffic. *wsHttpBinding* provides powerful security features, making it the binding type of choice for Internet communications. *wsHttpBinding* does not support streaming or duplex communications.
- **WSDualHttpBinding** Like *wsHttpBinding*, except it provides duplex communications when the service needs to initiate communications to a client. As discussed in Objective 1.1, duplex communications will not work if the client is behind a firewall or Network Address Translation (NAT) device.
- **basicHttpBinding** Like *wsHttpBinding*, *basicHttpBinding* is SOAP-based. However, it is based on earlier SOAP 1.1 standards and does not include the full set of *wsHttpBinding* features, such as encryption. *basicHttpBinding* is primarily useful for communicating with WS-Basic Profile conformant web services, such as ASMX-based web services.
- **webHttpBinding** A REST-style binding that functions differently than SOAP. REST uses a wider variety of HTTP commands than SOAP, such as *GET*, *PUT*, and *DELETE*.

If you don't need to communicate across firewalls or the public Internet, and all hosts are .NET Framework–based, you can choose from these more powerful bindings:

- **netNamedPipeBinding** The preferred binding type for communications between processes on a single computer.
- **netTcpBinding** The most powerful binding type when all hosts are based on the .NET Framework.
- **NetMsmqBinding** Useful when you need to queue messages for later processing. For example, the client might need to submit a task to a server that will not be able to process the message in a timely manner or is completely offline.
- **NetPeerTcpBinding** Provides peer-to-peer communications when more than two hosts are involved.

Use the flowchart in Figure 1-4 to choose a binding type for your scenario. Although the flowchart does not include all binding types, it does cover the most common uses.

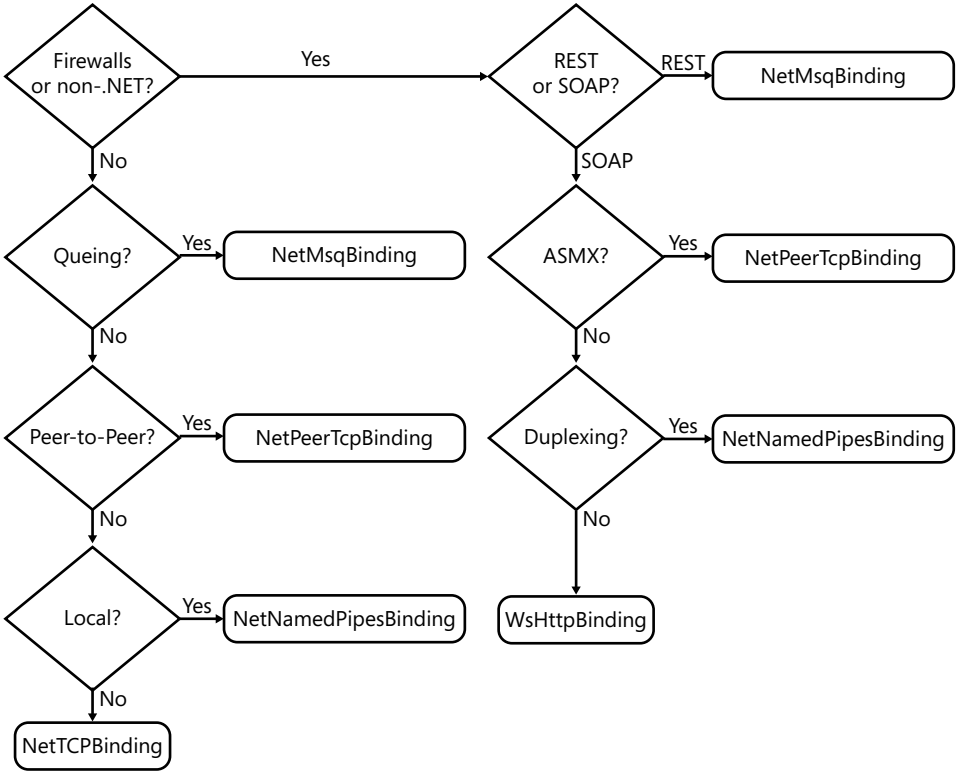


FIGURE 1-4 WCF binding decision flowchart

Any service you expose can use multiple bindings. Therefore, you can provide a binding based on *netTcpBinding* for .NET Framework–based hosts and a second *wsHttpBinding* for hosts that use open standards.

Although choosing the binding type is an important decision, it's relatively easy to change after the fact. Choose to define both the client and server bindings using configuration files, rather than hard-coding them into your application. Rely on discover protocols, such as Web Service Definition Language (WSDL), to save yourself from reconfiguring clients if the server settings change.

> ***MORE INFO*** **DESIGNING DATA ACCESS**
>
> **For information about communicating with a database, refer to Chapter 3, "Designing Data Strategies and Structures."**

When designing interactions between applications, consider whether your interactions will be chatty or chunky:

- Chatty application communications create many small requests. Chatty applications perform well when latency (the time it takes to send a message across the network) is low.
- Chunky application communications create fewer large requests. Chunky applications perform well when bandwidth (the total amount of data that can be sent across the network) is high.

Choose to bundle your communications together into chunks when the network infrastructure is high-latency and high-bandwidth, such as satellite or inter-continental communications. Use chatty communications by sending smaller messages immediately when the network infrastructure is low-latency and low-bandwidth, such as communications with mobile devices. The difference will be insignificant on most local area networks (LANs), which are low-latency and high-bandwidth.

Mapping the Logical Design to the Physical Implementation

After you have divided your application into layers to create your application's logical architecture, you need to design the physical architecture. The physical architecture defines the number of servers you will host the application on and how they are interconnected. At a high level, there are two different physical architecture philosophies:

- **Separate everything** Place the web server, application server, and database servers on different computers. This approach provides greater scalability.
- **Combine everything** Place all services on a single server. For the same cost, this approach provides better performance, reliability, and manageability.

Having a multitier logical architecture does not require you to deploy the same physical architecture. As Figure 1-5 illustrates, you can deploy all three logical layers to a single physical computer. Alternatively, you can deploy each layer to its own computer. You also have the option of deploying two layers to a single server, or of deploying ten web servers, three application servers, and a database cluster—the combinations are endless.

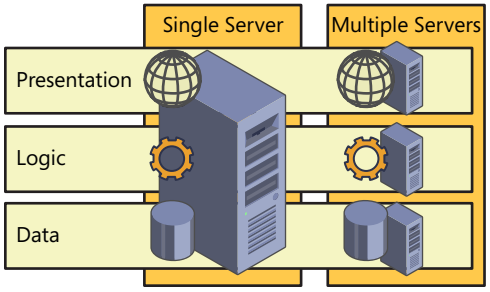


FIGURE 1-5 Single server vs. multiple servers for a three-tier web application

The physical architecture you choose depends on several factors. The single-server architecture has the following benefits:

- **Up-front costs** Obviously, if you deploy an application to three servers, you have to buy more hardware. This factor can be eliminated by using virtualization software, such as Hyper-V, and creating virtual machines for each server instance. You also need more operating system licenses. However, you might be able to use a less-expensive operating system for the web server, such as Windows Server 2008 R2 Web Edition. Instead of dividing your hardware and software budget across multiple servers, you can put the same money into a single server and achieve much better performance.
- **Management costs** The more computers you have, the more you need to manage. Over time, management costs typically exceed up-front costs.
- **Complexity** Simple is always superior. The more servers you add, the more complex your configuration will be.
- **Reliability** Each server you add to a web application is another component that can fail. If you deploy a three-tier application to three different servers, each server becomes a single point of failure; a failure of any one of the three servers would take the application offline. Similarly, scheduled downtime for any of the three servers requires scheduled downtime for the entire application. Additionally, communications between the servers becomes another single point of failure. If the network link between the servers fails, the entire application fails. You can use redundancy to overcome the decreased reliability of a multiple-server architecture, but your up-front and management costs will go up proportionately.
- **Efficient interprocess communications** Communicating across a network, even a LAN, always introduces some latency. It will always be faster for the web, logic, and data tiers to communicate when they are located on the same server.

The multiple-server architecture provides these benefits:

- **Scalability** With modern computing power, all but the busiest public web applications can be hosted on a single server. If your application requires more processing time or memory than a single computer can provide, dividing different tiers between physical computers can alleviate that concern.
- **Isolation** Deploying a tier to a separate physical computers guarantees that the load on other tiers will not affect that tier's performance. For example, if you are concerned that a denial-of-service attack might cause IIS to consume 100 percent of the server's processing time, you can place the database on a separate physical computer to prevent the database from being affected. The hosted application is still inaccessible because of the load on the web server, but if other applications share the same database, those applications are not affected (unless, of course, the denial-of-service attack caused the website to submit costly database queries).

EXAM TIP

For the exam, know that you can move a service (such as a database server) to a different physical computer if you absolutely must ensure that load on the web server never affects the service.

Real World

I've had the opportunity to design the architecture for hundreds of websites, ranging from startups to Fortune 100 companies, and then monitor the performance of the websites over years. About 80 percent of the time, the clients think they need far more hardware then they ever end up using. Even during peak hours, most applications running on a single physical server are under 2 percent utilized.

There's the argument that websites must be designed to scale to sudden spikes in popularity, such as being featured in the news. That's a valid concern, but I typically recommend deploying a single-server architecture and then using load testing (for example, using the Microsoft Web Capacity Analysis Tool) to determine whether the architecture can scale. In most cases, the limiting factor to scalability is not insufficient processor or memory. Instead, applications tend to run into artificial connection limits, resource locking problems, and other software-related issues.

When those issues are resolved, further load testing tends to show developers that they need to make better use of .NET Framework caching, which can almost completely eliminate web and database server processing requirements. If the application is well designed and coded, your upstream bandwidth will limit your performance—a problem that can be resolved by caching content (especially multimedia content) by using a distributed content delivery network (CDN), such as Akamai or Amazon CloudFront.

Objective 6.3, "Plan for Scalability and Reliability," in Chapter 6, "Designing a Deployment Strategy," provides more detail about how to meet quality of service (QoS) requirements.

Validating Nonfunctional Requirements and Cross-Cutting Concerns

Cross-cutting concerns and nonfunctional requirements are noncore functions that affect many parts of an application. For example, management, monitoring, logging, and authentication affect all parts of the application, but they are not part of the core functionality.

MORE INFO **OTHER CROSS-CUTTING CONCERNS**

This section focuses on operational cross-cutting concerns, because other concerns are covered by different exam objectives. For information about operational security, refer to Chapter 4, "Designing Security Architecture and Implementation." For information about performance monitoring, refer to Chapter 5, "Preparing For and Investigating Application Issues." For information about health monitoring, refer to Chapter 6.

As a developer, you should strive to create applications that systems administrators can configure and manage without needing development experience. You can do this by following these best practices:

- **Read all configuration settings from XML files, such as the Web.config file** You can do this by using the *ConfigurationManager.AppSettings* property. Administrators should already be familiar with editing XML files, so storing commonly changed settings (such as the address of a server or number of seconds in a timeout) in the Web.config file allows administrators to change an application's behavior without contacting the developer.
- **Store application settings in an external .config file** To allow an upgrade or re-installation to overwrite the Web.config file without affecting the application configuration, store application-specific settings in a separate XML file. You can reference this file by using the following syntax:

```
<configuration>
    <appSettings file="externalSettings.config"/>
</configuration>
```

- **Never store constants in a code file** It might seem obvious not to store connection strings in a code file; however, you should also avoid storing the number of rows that appear on a page, the number of retries, or the path to a shared folder in a code file.
- **Use resource files** It's often OK to store text, such as "OK" or "Cancel", in an .aspx file. However, you should never store text that appears in a user interface in a code file. Instead, reference a resource file that administrators can edit without recompiling the application.
- **Disable debugging by default, but allow it to be re-enabled** Ideally, if a systems administrator cannot solve a problem, she describes it to the developer, who then re-creates the problem in a lab environment. In practice, however, some problems cannot easily be re-created, and you have no choice but to debug an application in the production environment. Plan for this by designing applications that allow debugging with proper authentication.
- **Allow back-ups** Generally, web applications can be backed up as regular files. However, you must ensure that the data store you use can be backed up. If you use a SQL Server database, ensure that administrators know which tables to back up, and

how frequently to back them up. Avoid keeping files locked, which might prevent them from being backed up.

- **Start correctly after a reboot** Operations need to regularly restart servers. Although most web applications start automatically when the server comes back online, you should test your application to ensure that it functions properly after restarting either the web server, the database server, or both. In particular, avoid establishing database connections or reading important data in the *Application.Start* event, because the database server might be offline when the web server starts. A related trait to plan for is known as *graceful degradation* or *resiliency*, which allows the application to recover from a temporary network, database, or application server outage.
- **Plan for database changes** Table structure can change over time. For example, a future version of an application might add a column to a table. To provide forward-compatibility and ensure that different versions of your application can interact with a single database, refer to table columns using names rather than column numbers. This is handled automatically if you use Entity Data Modeling.
- **Document thoroughly** Naturally, you should comment both your code and configuration files. Additionally, you should write documentation for testing, deploying, configuring, monitoring, backing up, and restoring your application.

When you actually develop the application, avoid mixing functional and nonfunctional code. Instead, use the principals of aspect-oriented programming (AOP) and separate cross-cutting code into separate concerns.

MORE INFO **ASPECT-ORIENTED PROGRAMMING**

For more information, read "Aspect-Oriented Programming" at *http://msdn.microsoft.com/library/aa288717.aspx*.

Evaluating Baseline Needs

During the design phase, you need to evaluate the baseline needs of your web application. Don't look too far into the future; instead, focus on what the application will need during the first six months. When the application is in production, you will be able to examine real-world performance factors and plan more accurately.

The following list describes the most important information you need to gather:

- **Uptime requirements** These requirements are often expressed as 99% ("two nines"), 99.9% ("three nines"), or 99.99% ("four nines"). You need to determine whether a single server can meet the uptime requirements, factoring in planned downtime caused by updates.
- **Responsiveness** How long users are willing to wait for a response from your server. With Internet applications, you must also factor in the network latency.

- **Peak number of simultaneous users** The number of users who will be logged on simultaneously.
- **Peak number of requests per second** The number of requests the web application will receive within a second. If this is less than one (as it is with most new web applications), you typically don't have to worry about processor capabilities, as even low-end shared web servers will be able to process most requests in less than a second.

> ***MORE INFO*** **RELIABILITY AND SCALABILITY**
>
> **For more information about planning for scalability and reliability, refer to Chapter 6.**

Estimating these values allows you to create baseline infrastructure requirements, such as the following:

- **Disk capacity** .NET Framework code typically takes very little disk space; however, databases can grow very large. Provide at least twice the estimated capacity of your data to allow for the inherent inefficiencies of database storage.
- **Number of processor cores** Provide enough processing power to render pages faster than the peak number of requests per second. If the server cannot keep up with this number, the web server will queue requests. Short-term queuing might still occur when an abnormally high number of requests arrives within a few seconds, but the temporarily reduced responsiveness is typically acceptable.
- **Memory** The amount of random access memory (RAM) your application will store. If you plan to store any large collections in memory rather than accessing them from a database, ensure the web server has at least twice that space available. The web server will use any excess memory for caching.
- **Bandwidth** You can estimate the bandwidth requirements by multiplying the peak number of requests per second by the average page size (including images and video). Rendered webpages consume very little bandwidth. Instead, most web bandwidth is consumed by transmitting images and video.
- **Number of servers** You might need multiple servers to provide redundancy to meet the uptime requirements. If you are planning for a high peak number of requests per second and a single server cannot provide the processing power, you might need multiple servers to meet your responsiveness requirement.
- **CDN needs** For Internet applications, verify that your Internet connection has sufficient available bandwidth to meet peak requirements. If it is insufficient, use a web hosting provider with sufficient bandwidth or distribute the images and video by using a CDN.

Objective Summary

- The two most commonly used system architectures are MVC (discussed in Objective 1.1) and three-tier. The three-tier architecture creates an SoC between the presentation, application logic, and data tiers.
- Unless you are required to work with a system that does not support it, you should use MFC to communicate between applications. MFC provides a wide variety of protocols to meet interoperability, performance, and queuing requirements.
- When mapping logical design to physical implementation, use the minimum amount of servers required to meet your needs. Unless you need multiple servers for redundancy, scalability, or isolation, use a single server.
- Cross-cutting concerns such as monitoring, logging, operations, and security are not related to the application's core functionality. However, they contribute greatly to the manageability of the application. Whenever possible, use coding best practices that separate cross-cutting concerns into separate classes and store settings administrators might want to change separately from your code.
- Although baseline needs must be estimated, they are important because you can use them to specify hardware requirements. Typically, you should err on the side of using less expensive hardware, but provide a convenient path to upgrade in the event the application has higher requirements after it is in production.

Objective Review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You need to design a physical architecture to meet these requirements:
   - The website will remain online if the web server is restarted.
   - The database server will not be affected by a denial-of-service attack against the web server.
   - You must minimize hardware costs.

   How many servers will you need?

   A. One

   B. Two

   C. Three

   D. Four

2. You need to design a physical architecture to meet these requirements:
   - The website must be able to serve six requests per minute.
   - The database server will store 2 TB of data.
   - You must minimize hardware costs.

   How many servers will you need?

   A. One

   B. Two

   C. Three

   D. Four

3. You are designing a three-tier web application. You need to choose the method of communication between the web and application layers to meet these requirements:
   - The application server will use the .NET Framework 4.0.
   - The application server must be physically isolated.
   - The application server will be located on the same high-speed LAN as the web server and database servers.
   - The web server will be located behind a firewall.
   - The communications will be two-way.
   - The communications must be as efficient as possible.

   Which WCF binding type will you use?

   A. *NetTcpBinding*

   B. *NetNamedPipesBinding*

   C. *WsHttpBinding*

   D. *WSDualHttpBinding*

THOUGHT EXPERIMENT
Planning for Scalability and Forward Compatibility

In the following thought experiment, you apply what you've learned about the "Analyze Requirements and Recommend a System Topology" objective to predict how a theoretical website architecture will perform. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a developer for Margie's Travel, an Internet-based business that provides travel reviews and recommendations. Margie's Travel has hired a developer to create an updated version of its website, and the company has asked you to review the design to verify that it meets its requirements for scalability and forward compatibility.

The new application is designed with a traditional three-tier architecture, as shown in Figure 1-6. Initially, all three layers will be implemented using a single server.

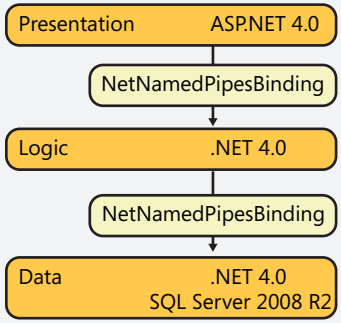


FIGURE 1-6 A three-tier application for Margie's Travel

Management wants to verify that the design meets its requirements. Answer the following question about the future performance of the application:

1. **What changes will the company need to make to the application if it moves each tier to its own server?**
2. **What changes will the company need to make to the logic tier if it wants to create a Windows Presentation Foundation (WPF) application as a secondary presentation interface?**
3. **What changes will the company need to make if it replaces SQL Server 2008 R2 with a non-Microsoft database server?**
4. **What changes will the company need to make if it replaces the ASP.NET presentation interface with a Linux presentation interface? What impact will that have?**
5. **The company expects the website to get about 200 visitors per day. Will a single, dedicated server be fast enough?**

Objective 1.3: Choose Appropriate Client-Side Technologies

Client-side technologies can make webpages feel responsive and interactive. However, because they run on the client, they can be challenging to plan. The richest client-side technologies require plug-ins, which some browser platforms do not support. Although JavaScript has become standard, many developers use client-side libraries that can increase bandwidth and page load times.

This section provides an overview of common client-side scripting languages and plug-ins, and it gives you the information you need to choose the right technology for different scenarios.

This objective covers how to:

- Use client-side scripting languages.
- Use rich client-side plug-ins.

Using Client-Side Scripting Languages

In practice, there is only one client-side scripting language: JavaScript. The JavaScript syntax closely resembles C# syntax because it requires lines to end with a semicolon and uses brackets to group lines of code together (for example, to group the code that makes up a function or a for loop). However, the structure is looser than C#, and you don't always need to declare variables.

Here is a sample of JavaScript code to demonstrate the syntax. Because anyone with access to a page can examine the source code, JavaScript code rarely contains comments in the real world. To reduce bandwidth usage, JavaScript is almost always minified, which means unnecessary indenting and white space has been removed.

```
// Declare variables (without specifying a type)
numPics = 11;
secDelay = 7;

// Call the built-in functions Math.floor and Math.random
// Adding "var" identifies a local variable
var randomOffset = Math.floor(Math.random() * 11);

// A for loop
for (i = 1; i <= numPics; i++) {
    // Declare and define picNum
    picNum = i + randomOffset;

    // An if statement
    if (picNum > numPics) {
```

```
        picNum = picNum - numPics;
    }

    // Define a function call that will be passed to setTimeout using a string
    // StartPic is a custom function not shown here
    var sp = "StartPic(" + picNum + ", " + numPics + ", " + secDelay + ", '" +
        picUrl[picNum] + "')";

    // setTimeout, a built-in function, runs the function declared by the first
    // parameter after the delay, in milliseconds, specified by the second parameter
    setTimeout(sp, secDelay * 1000 * (i - 1));
}
```

EXAM TIP

On the exam, you might see a reference to a second client-side scripting language: Microsoft VBScript. VBScript provides similar client-side capabilities as JavaScript; however, VBScript is supported only by the Microsoft Internet Explorer browser. Although most users currently use a version of Internet Explorer, most web developers prefer to write code that will work with as many browsers as possible, including Firefox, Opera, Chrome, and Safari. Only JavaScript works with each of those browsers. In the real world, almost all client-side code is written in JavaScript.

Microsoft is still maintaining VBScript; however, it is no longer releasing new versions of the scripting engine. For the exam, it is important to know that VBScript is a client-side scripting language with similar capabilities to JavaScript. However, VBScript is not an acceptable solution for scenarios that require compatibility with browsers other than Internet Explorer or operating systems other than Windows.

Client-Side Libraries

JavaScript itself is not as robust as the .NET Framework. For example, it could be time-consuming to write pure JavaScript code to retrieve data from a web service, sort it, and display it in a grid format, because JavaScript does not have built-in functions to do that. Even if you did write the large amount of code required to perform that task, you would then have to test and debug it using every different browser you planned to support.

To extend JavaScript's capabilities and provide better cross-platform support, the open-source community has created many JavaScript libraries. There are two you should be familiar with for the exam:

- **jQuery** jQuery is the most commonly used JavaScript library. It is only 29 KB in size when compressed (which the client must download before it can process any JavaScript), but it provides functions for easily selecting document elements, handling events, animating objects, and performing other AJAX actions.
- **Microsoft AJAX** A client-side JavaScript library created by Microsoft specifically to add client-side capabilities to ASP.NET.

Both jQuery and Microsoft AJAX are included with Visual Studio 2010. However, new versions of each are released regularly. Because they often contain important updates, you should always use the latest version available. Additionally, you should test and update references to new libraries for web applications you maintain.

> ***MORE INFO*** **JQUERY AND MICROSOFT AJAX**
>
> **For more information about jQuery, visit *http://jquery.com*. For more information about Microsoft AJAX, visit *http://www.asp.net/ajax*.**

Using a client-side library on your page requires the client to download the library from your server the first time a page requests it. For all subsequent requests, the client will normally have a copy cached. Unfortunately, the initial download of the library increases page load time and bandwidth usage. On a LAN, these differences might be negligible. On the public Internet, however, they can be significant.

For example, the jQuery library is only 29 KB in size when minified, which broadband users can download in a fraction of a second. However, the browser has to first download the page, parse the reference to the jQuery library, download jQuery from the server, and then process the library before it can begin executing any JavaScript. Depending on the client's bandwidth and latency (the delay it takes to send messages to and from your server), that can delay page-rendering time by up to a full second.

Because of the performance impact of loading client-side libraries and the importance of page load time to users and search engines, avoid using libraries for websites on the public Internet unless absolutely necessary. At times, however, using a library can save such a significant amount of development time that the performance impact becomes worthwhile.

Delivering Libraries with a CDN

To reduce the impact of downloading a library, you can use a CDN. A CDN stores copies of a library in many different locations on the Internet. Browsers download whichever copy of the library is closest to them, reducing latency and eliminating the extra bandwidth required of your web server. Additionally, if the browser has visited another webpage that uses the same version of the library from the same CDN, it can use the cached version, further improving page-load time.

Microsoft, Google, and Edgecast all provide CDN services for the jQuery library. Instead of referencing jQuery from your local server:

```
<script type="text/javascript" src="/Scripts/jquery/jquery-1.5.min.js" ></script>
```

you can reference it from Microsoft's CDN:

```
<script type="text/javascript" src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.5.min.js" ></script>
```

you can reference it from Google's CDN:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/
jquery.min.js"></script>
```

or you can reference it from Edgecast's CDN:

```
<script type="text/javascript" src="http://code.jquery.com/jquery-1.5.min.js"></script>
```

> ***NOTE*** **JQUERY VERSIONS**
>
> **The .min in the file indicates that it is minified, which reduces file size. You need to reference the nonminified version only if you wanted to examine the jQuery library for debugging purposes. As developers release new versions of jQuery, you should test it and then update the version number in the script *src* property.**

If a user has visited another page that used the same library from the same CDN, the browser will already have a cached copy of the CDN stored locally. Therefore, the more webpages that use a CDN, the more visitors will have already cached the library. As a result, you will get a greater performance benefit by using the most popular CDN.

To use the Microsoft CDN for Microsoft AJAX (your only option), simply set the *ScriptManager.EnableCdn* property to *true*, as the following example shows:

```
<asp:ScriptManager
    ID="ScriptManager1"
    EnableCdn="true"
    Runat="Server" />
```

Using Rich Client-Side Plug-ins

JavaScript is capable of providing a moderately rich user interface that changes based on user actions. JavaScript can also display some video and animation. However, if you want to provide a rich user interface beyond JavaScript's capabilities, use high-performance or 3D graphics, or display some types of video, you need to use a plug-in. Two of the most common plug-ins are Adobe Flash and Microsoft Silverlight.

Flash and Silverlight are capable of complex, interactive animations, high-definition video, and games. They can also directly access some aspects of the computer that are not accessible to JavaScript, such as the video adapter's graphical processing unit (GPU), which is useful for accelerating graphics displays and performing some computations. However, Flash and Silverlight have several disadvantages:

- They require users to download and install a plug-in before they can be used.
- Users who already have the plug-in installed might have to update the plug-in before they can access content created for a newer version of the plug-in.
- Some browsers (such as the Safari browser built into the popular iPhone and iPad products) cannot support one or both of the plug-ins.

- The Flash and Silverlight objects must be stored in a separate file and embedded in a webpage. These objects tend to be large, increasing the size of the page and reducing page performance.

Because of these drawbacks, you should use plug-ins only when you can limit the impact of the drawbacks. For example, if you have a technology-savvy customer base that is willing to install and update the plug-in, or if you are deploying an intranet application and the Information Technology (IT) department will support the plug-in.

If you create web applications that use Flash or Silverlight and you must support a wide variety of browsers, create webpages that degrade gracefully. If a browser does not have the plug-in installed, provide a link so that users can install the plug-in. If a browser cannot support the plug-in, provide alternate content that duplicates as much of the functionality as possible using traditional HTML and JavaScript.

EXAM TIP

In the real world, Flash is much more widely supported than Silverlight because it has existed for many more years. The exam, however, prefers to reference Silverlight. Fortunately, you do not need to know how to create content for either plug-in. Instead, understand the capabilities and drawbacks of using rich client-side plug-ins.

Objective Summary

- JavaScript is the only widely accepted client-side scripting language, though Internet Explorer also supports VBScript. You can use client-side libraries to extend JavaScript's capabilities. Visual Studio 2010 has built-in support for two client-side libraries: jQuery and Microsoft AJAX.
- Rich client-side plug-ins, including Flash and Silverlight, provide a responsive and interactive experience rivaled only by desktop applications. However, they both require plug-ins to be installed in most browsers. Some browsers, especially those on mobile devices, do not support either plug-in. Therefore, when using a rich client-side plug-in, you need to plan to provide alternate code for clients that lack the plug-in.

Objective Review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are designing an ASP.NET web application that allows end users to chat live with customer support when they click a Button control. At times, it might take several minutes for customer support to respond to a user. You need to design the application to be responsive to users, even when customer support cannot immediately reply. You need to support a wide variety of browsers, including browsers built into mobile devices. Which approach should you recommend?

   A. In the *Button.Click* event handler, return a webpage that contains a real-time chat client implemented by using Flash. Configure the Silverlight client to communicate directly with customer support.

   B. In the *Button.Click* event handler, return a webpage that contains a real-time chat client implemented by using Silverlight. Configure the Silverlight client to communicate directly with customer support.

   C. In the *Button.Click* event handler, connect to the customer support chat client. Add the response to the webpage.

   D. In the *Button.Click* event handler, return a webpage that contains client-side JavaScript code that connects to a web service running on the server to send messages from the user and retrieve responses from customer support.

2. You are designing the client-side component of web application with the following requirements:

   - Retrieve data from a web service, and display it in a element on the webpage.
   - Work with all common browsers.
   - Do not require client-side installation.

   Which two technologies should you recommend? (Choose all that apply. Each answer forms a complete solution.)

   A. Silverlight

   B. jQuery

   C. VBScript

   D. Microsoft AJAX

3. You are designing a 3D game that will run in client browsers. The platform you choose must be able to work in all common browsers and must support GPU acceleration. Which technology should you choose?

   A. Silverlight

   B. jQuery

   C. VBScript

   D. Microsoft AJAX

THOUGHT EXPERIMENT
Evaluating the Impact of Rich Client Features

In the following thought experiment, you apply what you've learned about the "Choose Appropriate Client-Side Technologies" objective to predict how a theoretical website architecture will perform. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a development consultant. Coho Winery has asked you to review an MVC web application that its developer created to replace its public website at http://www.cohowinery.com. Their current public website is based on ASP.NET 2.0 and uses no JavaScript or other client-side technology. Management needs you to evaluate the application design to identify any problems that might occur prior to launch.

You interview the developer and determine that three components of the site use advanced client-side capabilities:

- The master page has a menu system that uses jQuery. jQuery is stored in the application's /Scripts/ folder.
- Product pages use AJAX to allow users to quickly browse different products without reloading the entire page.
- The home page displays a 500 KB Silverlight object that gives users a virtual tour of the winery.

Management needs to assess the future performance and compatibility of the site. In particular, they are concerned about bandwidth fees from their web hosting provider. Answer the following questions:

1. Will their bandwidth usage increase or decrease?
2. How can they change the design to reduce bandwidth usage?
3. Will any clients be unable to use parts of the site? How can they minimize the compatibility problems?

Objective 1.4: Choose Appropriate Server-Side Technologies

The .NET Framework is so robust that there are often many ways to accomplish a single task. This objective includes a high-level overview of the different server-side technologies and describes the scenarios in which you would use each of them.

This objective covers how to:

- Choose between different control types.
- Use partial classes and methods.
- Access server methods from client code.

MORE INFO **HTML Helper Extensions**

For information about HTML helper extensions, refer to Objective 3.2.

Choosing Between Different Control Types

The .NET Framework provides many types of controls, including HTML, server, user, Web Parts, custom, and dynamic data. The sections that follow provide an overview of each type of control and describe the scenarios in which you would use each.

EXAM TIP

The 70-515 exam covers how to create and use controls. For the 70-519 exam, be sure you understand when you should choose each type of control.

HTML Controls

Use HTML controls when you need to create an HTML element without server-side logic. For example, you can create an input text box using either an HTML control or a server control. The Input (Text) HTML control simply creates the HTML <input> element, without directly providing server-side code access to anything the user types in the text box. If the user submits a form to the server, the contents of any HTML controls will be lost. You could still access the user input in an HTML control from client-side JavaScript, however.

Server Controls

ASP.NET renders server controls into HTML elements. For example, the *TextBox* server control is rendered into an HTML <input> element, similar to the Input (Text) HTML control. Server controls provide more robust features, however. With server controls, you can process user

input from the code-behind file, and ASP.NET automatically maintains controls between page loads. Server controls require more processing time on the server, and their view state consumes additional bandwidth. As with HTML controls, you can access rendered server controls from client-side JavaScript.

Although the 70-519 exam does not test most of the details of implementing server controls, it does require you to know how to enable Cascading Style Sheet (CSS) styling for controls that have it disabled by default. These server controls were originally designed before CSS was commonly used, and as a result, they use tables for styling. All modern web applications should use CSS styling.

The .NET Framework provides different properties for different controls:

- **RenderOuterTable** Several server controls (including *FormView*, *Login*, and *Change Password*) have a property named *RenderOuterTable*. Set the *RenderOuterTable* property to *false* to control the appearance of the controls with CSS style sheets.
- **RepeatLayout** *RadioButtonList*, *CheckBoxList*, and *DataList* include the *RepeatLayout* property. Like *RenderOuterTable*, *RepeatLayout* uses a table for formatting. Set *Repeat Layout* to Flow (instead of the default Table) to control the appearance of the controls with CSS style sheets.

User Controls

User controls are custom server-side controls deployed using an .ascx file. Typically, user controls contain multiple server or HTML controls. You should create user controls under the following circumstances:

- You need to include a group of server controls in multiple ASPX pages.
- You need to deploy a custom control as a separate assembly.
- The layout is primarily static.

If you need to persist data between page loads, store the data in control state. Control state functions even when a developer has disabled view state.

Do not use user controls to display common user interface elements in the same position on every page within your site. Instead, use master pages. For example, if you want to display the weather in the upper-right corner of every page, you should add the weather element to your master page. However, if you want to display the weather in different locations on different pages, you should implement it as a user control.

Web Parts

Web Parts are controls that include built-in functionality to allow end users to customize their location, appearance, and functionality. You can use Web Part Connections to provide data to individual Web Parts. Choose Web Parts over other types of controls when user customization is important—for example, if you need the user to be able to move the control to a different spot on the webpage or change the colors of the control.

Custom Server Controls

Although user controls are typically composed of HTML and server controls, custom server controls provide you complete flexibility. You have complete control over how ASP.NET renders a custom server control into HTML, so you can accomplish almost anything. Custom server controls take more effort to create than user controls, however.

If your custom server control does not resemble any existing server controls, inherit from *System.Web.WebControls.WebControl*. Most of the time, however, you should inherit from an existing server control that provides similar functionality to your custom server control. For example, if you wanted to display a list of check boxes with associated images, you might inherit from the *CheckBoxList* class; add properties for the image URL, title, and alt tags; and then update the rendering to add the <img> information to the HTML output.

You can add custom controls to the Visual Studio toolbox, which you cannot do with user controls.

Dynamic Data Controls

Dynamic data controls are among the most complex controls in the .NET Framework. Dynamic data controls connect to a LINQ-to-SQL object model or LINQ-to-Entities object model and display data in a tabular format, allowing for sorting, filtering, and paging. Users can use links to edit and delete records directly from a dynamic data control.

Choose dynamic data controls when you want to view or edit data in a database with minimal development time and a standard tabular display will suffice.

Using Partial Classes and Methods

When you create a new website or web application, Visual Studio automatically generates a large amount of code. This auto-generated code includes many of the most important classes in your application, including your *Page* class and your data classes. By automatically generating code based on the template you chose, Visual Studio saves you from writing hundreds of lines of code just to provide basic website functionality.

You can use partial classes and methods to add your own custom code to the auto-generated code. A partial class allows you to add entirely new properties, methods, and events to a class, without editing the auto-generated code or rewriting the class from scratch. Similarly, a partial method allows you to add code to an auto-generated method.

Use partial classes and methods when Visual Studio auto-generates a class or method but you need to extend its functionality.

Accessing Server Methods from Client Code

The most common way to call a server-side method from the client is to configure a server control, such as a *Button* control, to run the method for its *Click* event handler. However, this technique requires a postback.

If you prefer not to reload the page, you can use one of these techniques to allow client-side code to call server-side methods and process the results:

- **UpdatePanel** You can place server controls within an *UpdatePanel* container, and trigger the *UpdatePanel* to refresh itself when a trigger occurs. Often, triggers are clicking a button or changing a drop-down list selection. Using an *UpdatePanel* does not require manually writing JavaScript code.
- **UpdateProgress** *UpdateProgress* is visible only when an associated *UpdatePanel* control is updating. *UpdateProgress* is designed to give users feedback about the *UpdatePanel* asynchronous request.
- **Page methods** Mark the static method you want to expose with the *WebMethod* attribute, add a *ScriptManager* server control, and set *ScriptManager.EnablePageMethods* to *true*. Then you can use the *PageMethods* object within JavaScript to call server-side methods and process the results.
- **Web services** Microsoft AJAX and jQuery are both capable of easily consuming web services. Therefore, you can write client-side code to consume any static method that you expose by using *WebMethod*. To access the method using the JSON format, also add the *ScriptService* attribute.

Using an *UpdatePanel* requires less code than using a page method, but it transfers view state as well, so the request and response are larger than page method calls. With a page method control, you can't access the contents of server controls, such as what a user typed into a *TextBox*, directly from a page method called by the client. Instead, you need to pass values to the page method as parameters.

Objective Summary

- HTML controls provide high performance with minimal overhead, but server controls provide much easier access to their properties and values. User controls allow you to easily combine multiple server controls into a single object. Web Parts provide robust capabilities and allow end users to personalize their appearance and move them to different locations. For the ultimate in flexibility, create custom server controls. Dynamic data controls automatically adjust their appearance to the underlying data source, allowing you to quickly display and manage data.
- Partial classes and methods allow you to extend classes that Visual Studio automatically generates.
- HtmlHelpers provide an easy way to add HTML controls to MVC views. You can create custom HtmlHelper extensions for HTML elements that are not provided with the built-in HtmlHelpers.
- You can access server methods directly from client JavaScript code, saving a full page load. The easiest way to do this is to use a page method with Microsoft AJAX.

Objective Review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are creating a new web application. You must meet these requirements:
   - Use the *Login* control to allow users to authenticate.
   - Allow a web designer to configure the appearance of all controls using classes in CSS style sheets.

   How should you configure the *Login* control? (Choose two. Each answer forms part of the complete solution.)

   A. Define the CssClass property.

   B. Set the *RenderOuterTable* property to *true*.

   C. Set the *RenderOuterTable* property to *false*.

   D. Define the *LoginButtonType* property.

2. You are planning to write client-side JavaScript that must retrieve and display a string value returned by a server-side method. You want the messages sent between the client and server to be as small as possible. Which approach should you choose?

   A. Create a partial class.

   B. Use an *UpdateProgress* control.

   C. Use an *UpdatePanel* control.

   D. Use a Microsoft AJAX page method.

3. You are creating a view for an MVC application. You need to create a non-standard HTML control that should be rendered on the server. You must minimize the amount of client and server resources used as well as your development time. Which approach should you choose?

   A. Use the jQuery library to dynamically add the HTML control to the document object model.

   B. Create an HtmlHelper extension.

   C. Create a custom server control.

   D. Create a custom Web Part.

THOUGHT EXPERIMENT
Evaluating a Real-Time Web Application Design

In the following thought experiment, you apply what you've learned about the Choose Appropriate Server-Side Technologies objective to predict how a theoretical website architecture will perform. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a developer for Contoso Pharmaceuticals. Recently, Contoso acquired a smaller pharmaceutical company, Trey Research. The developers at Trey are in the process of planning a new web application. Your manager has asked you to review the Trey developer's plans.

The application is designed to provide management with real-time information about the progress of drug development, whether the manager is in the office, at home, or on a mobile device. To support mobile clients, one of the design requirements is to minimize bandwidth. Managers want to be able to open a webpage and leave it open, and have it automatically update with current project status.

Trey developers designed an application with the following features:

- A user control that displays all the information about a project
- An *UpdatePanel* control that contains one user control for every project that a manager wants to monitor
- A *Timer* control that triggers the *UpdatePanel* control to refresh every 30 seconds

Answer the following questions about the future performance of the application:

1. Is a user control the right choice for displaying information about a project? Which other technologies might be more efficient?
2. Is the combination of an *UpdatePanel* and *Timer* trigger the right choice to allow the page to be dynamically updated? Which other technologies might be more efficient?

Objective 1.5: Design State Management

As a user visits a website, he might interact with multiple controls on a page, submit a single page multiple times, and view multiple, different pages in a single site. Often, the user will return later to the same site.

Throughout these interactions, your application needs to keep track of state. You need to know any values the user types or selects in a form. Often, you need to track information about a user as he visits multiple pages or makes multiple visits to your site at different times.

Keeping track of this information in a web application is a challenge, however, because HTTP communications are inherently stateless, a web application might run on multiple web servers, and a single user might switch between different networks or clients. Fortunately, ASP.NET provides several state management techniques that can meet almost any need.

This objective includes an overview of those different state management techniques, along with a description of the advantages of each. Most of this objective reviews topics covered by the 70-515 exam. However, the 70-515 exam focused on the details of implementing state management technologies, while the 70-519 exam focuses on choosing the appropriate state management technology for a given scenario. Therefore, the content in this objective is very high level.

This objective covers how to:

- Use application state.
- Use the Cache object.
- Evaluate user state technologies
- Use session state.
- Create custom page state persisters.

Using Application State

ASP.NET provides the *Application* object, which you can use to store information that can be accessed by any page or user session. The *Application* object is a dictionary that is useful for storing small amounts of frequently accessed data that is the same for all users.

However, the *Application* object was primarily intended for backward compatibility with ASP applications. A better alternative is to store application state in global static objects. Global static objects perform better than the *Application* object and provide strong typing.

Because application state is stored in memory, accessing it is much faster than querying a database. Consider these factors when using application state:

- Application state is lost when a server is restarted. Therefore, you must initialize data in the *Application_Start* method in the Global.asax file.
- Application state is not shared between servers in a web farm. If you need to share data between servers, use session state, or store it in a shared database.

- Application state is stored in memory. If you store large amounts of data in application state, it reduces the amount of memory available for other applications and caching. Use the *Cache* object to allow the .NET Framework to automatically remove objects you no longer need from memory.
- Application state is free-threaded. Because multiple threads can access application state at the same time, you must write thread-safe code that locks and unlocks data so that it is written to by only one thread at a time. Writing thread-safe code increases development time.

Using the *Cache* Object

Just like the *Application* object, the *Cache* object is a dictionary that is available to all pages and sessions in your application. However, while *Application* stores objects until the application restarts, *Cache* stores them only until they expire or ASP.NET determines that it needs to free up memory.

When you add an object to the Cache, you have the option of providing an expiration policy. For example, the following code configures ASP.NET to remove the object after one minute:

Sample of Visual Basic.NET Code

```
Cache.Insert("MyItem", "MyValue", Nothing, DateTime.Now.AddMinutes(1.0), TimeSpan.Zero)
```

Sample of C# Code

```
Cache.Insert("MyItem", "MyValue", null, DateTime.Now.AddMinutes(1d),
    System.Web.Caching.Cache.NoSlidingExpiration);
```

Alternatively, you can create sliding expiration policies and policies that cause a cached object to expire when a file or directory is updated.

You should use the *Cache* object any time you might need to access the same data again and that data is relatively difficult to retrieve or create. When choosing between storing data in the *Cache* and *Application* objects, choose the *Cache* object in the following circumstances:

- The original data source might change.
- You might run low on memory.
- You will not need the value for the entire lifespan of the application.

Evaluating User State Technologies

User state is any information the server maintains between multiple user requests. ASP.NET provides several ways to store user state:

- **Cookies** Short strings provided by the server that the client includes with each subsequent request. You can store any string on the client and instruct the client to store it for a few seconds or many years, providing a simple and persistent storage technique. Because cookies do not store data on the server, they do not affect server scalabil-

ity. However, cookies provide no inherent security; they can be easily intercepted or modified. Additionally, because the client must send cookies with every request, large cookies can increase bandwidth usage and slow page response times.

- **Query strings** You can add query strings to hyperlinks on your pages to pass data to other pages. For example, if you link to a page that lists a store's inventory, you might specify the link like this to show only audiobooks under $10: http://www.contoso.com/products.aspx?category=audiobook&maxprice=10. Within the Products.aspx code-behind file, you could check for the values *Request.QueryString["category"]* and *Request.QueryString["maxprice"]*. Unlike other forms of application state, query strings are typically maintained if a user shares a link. Relying heavily on query strings can make it more difficult for search engines to index your site.
- **Hidden fields** You can store information within a form by using HTML hidden fields. If the user submits the form, the browser will submit the contents of the hidden field along with any other fields in the form. However, the hidden field won't be visible to the user. To add a hidden field, simply add an <input> element to the form and set the type to hidden, as this example shows:

```
<input type="hidden" name="code" value="93">
```

- **Session state** Clients identify themselves with a unique session ID, usually using a cookie. The server then retrieves a collection of session data using the user's unique session ID. Sessions expire after 20 minutes by default and will not be available if the user changes devices. Therefore, session state is useful only for short-term storage.
- **View state** ASP.NET stores view state in encrypted hidden fields in a web form. View state allows ASP.NET to track control values across multiple requests even if session state is disabled. View state can significantly increase page size in a way that cannot be compressed as efficiently as standard HTML. Therefore, you should disable view state when you do not need it.
- **Control state** Control state functions exactly like view state, except it is defined as part of a user control. If a developer disables view state for a page but a user control uses control state, that control state will remain intact.
- **Authentication** If you authenticate users, you can store any information about the user in a database and associate it with their user ID. This allows state to be persistent between visits, even if the user switches devices. However, users must log on to retrieve the state.

You should use cookies to store information about user preferences. Use view state and control state to store data between requests to the same web form on an intranet. Any data requiring security, such as a user's address, should be stored on a database and accessed only on the server after the user authenticates.

Table 1-4 compares the features of the different application-state storage techniques.

TABLE 1-4 Application-State Storage Techniques

Cookies	Query strings	Hidden fields	Session state	View state/ Control state	Authentication
Uses server resources					
			X	X	X
Can be shared in a link					
	X				
Can increase bandwidth significantly					
X		X		X	
Provides some security					
			X	X	X
Works across multiple views of the same page					
X	X	X	X	X	X
Works across different pages in a single visit					
X	X	X	X		X
Works across multiple visits from the same device					
X					X
Works across multiple visits from different devices					
					X

Using Session State

Session state involves both a client and server component:

- The client must identify itself to the server by using a unique session ID.
- The server must use the session ID to look up that client's unique Session collection.

The sections that follow discuss the decisions you can make about both the client side and the server side of session state.

Tracking Session on the Client

You have different options for both the client and server components. For the client, you can choose between using cookies or cookieless session state. Using cookies is the default setting, and it is the right choice for the vast majority of websites. Although every modern browser

supports cookies, you might be faced with a scenario that requires you to support sessions without cookies.

Cookieless session state appends the unique session ID to the webpage URL. For example, if a client's session ID is ow2j32ieo233kj4i2ogfj9320, the URL might be http://www.contoso.com/(S(ow2j32ieo233kj4i2ogfj9320))/default.aspx. The altered URL creates several problems:

- Search engines might receive a different session ID on different visits, causing the search engine to identify different paths to seemingly duplicate content.
- If a user bookmarks a page and visits it after the session has expired (20 minutes by default), the session ID will no longer be valid.
- The URL is much longer than normal, making it more difficult for users to share (especially using short message formats such as Twitter).
- If a user shares a URL with a session ID embedded, the other user can access information stored within her session. This creates a potential security risk.

Because of these concerns, you should choose cookie-based sessions (the default) unless clients cannot use cookies and you must rely on session state.

Storing Session on the Server

Your web server can store session state in one of three ways:

- **InProc** The default, this setting stores session state in the server's memory. Session state is not shared between different web servers, and it is lost if the web server is restarted.
- **State Server** This setting stores session state on a separate server. Multiple web servers can share state server session state, allowing clients to transparently switch between different web servers in a server farm. Session state is lost if the state server is restarted.
- **SQL Server** This setting stores session state on a computer running SQL Server. Like a state server, multiple web servers can share state server session state. Additionally, session state is maintained if the state server is restarted. If session state must be maintained during a server outage, you can store state in a SQL Server cluster.

InProc session state is sufficient for most applications that use a single web server. If you use multiple web servers (for example, as part of a network load-balancing cluster) and requests from a single client might be sent to a different server, you need to store state on a state server or SQL Server. In this scenario, you must define the same <machineKey> setting (located within <configuration><system.web> in the Web.config file) on every web server.

Because a machine key is automatically generated by default, you need to manually generate a machine key and set the value on every web server. You can use this online tool

to quickly generate a <machineKey> value: *http://aspnetresources.com/tools/machineKey*. The setting in the Web.config will resemble the following:

```
<configuration>
    <system.web>
        <machineKey
validationKey="8E8992656E0CD2811EA23ADA31DD7F75F199EE9476947E0860FFC9C767992AEDE0B5CFDAB
A73D059E67AB166491E1342E4101B814135CFE40BC51D55E4F6B4DE"
decryptionKey="8D02DDC2E9CE3647E5F2649DF5BA0F7A30CFAE2D5AE436AE809CA11D3A3F2121"
validation="SHA1" decryption="AES" />
    </system.web>
</configuration>
```

Creating Custom Page State Persisters

View state has a significant disadvantage: it can dramatically increase page size. This page size increase is especially dramatic when you are using complex controls such as *DataGrid*. At times, view state size can approach 1 MB, which takes about 15 seconds to transfer across a 512 Kbps link. You can offset this page size increase by disabling view state for controls that don't need to be persisted between requests, but often that is not sufficient.

View state is the default mechanism for persisting page state, and usually it is the best choice. However, it is not the only choice. View state uses the class *System.Web.UI.HIddenField PageStatePersister*, which derives from the *PageStatePersister* class. If view state does not meet your needs, you can use the other built-in page state mechanism, *SessionPageStatePersister*, or derive your own custom class from *PageStatePersister*.

SessionPageStatePersister stores view state data along with session state data. This eliminates the extra bandwidth consumed by sending view state data back and forth between the client and the server. However, it consumes more server resources by increasing session state size. Additionally, it can fail if you make use of iframes, pop-ups, or AJAX, or if users have multiple pages open simultaneously.

To use a different page state persister, register a custom *PageAdapter* class. Within the class, override the *GetStatePersister* method and return an instance of your *PageStatePersister*. Then use the custom *PageAdapter* class instead of the default page adapter. This example demonstrates how to use *SessionPageStatePersister*:

Sample of Visual Basic.NET Code

```
Public Class SessionPageAdapter
   Inherits System.Web.UI.Adapters.PageAdapter

   Public Overrides Function GetStatePersister() As PageStatePersister
      Return New SessionPageStatePersister(Page)
   End Function 'GetStatePersister

End Class 'SessionPageAdapter
```

Sample of C# Code

```
public class SessionPageAdapter : System.Web.UI.Adapters.PageAdapter {
    public override PageStatePersister GetStatePersister() {
        return new SessionPageStatePersister(Page);
    }
}
```

MORE INFO **PAGE STATE PERSISTERS**

For more information about storing page state in sessions, read "SessionPageStatePersister Class" at *http://msdn.microsoft.com/en-us/library/system.web.ui.sessionpagestatepersister.aspx*. For detailed information about creating a custom page state persister, read "PageStatePersister Class" at *http://msdn.microsoft.com/en-us/library/system.web.ui.pagestatepersister.aspx*.

Objective Summary

- The *Application* object provides a dictionary that can be accessed by any page and session. Use it sparingly because any object you add remains in memory until the application restarts.
- The *Cache* object provides a dictionary that can be accessed by any page and session, just like the *Application* object. However, the *Cache* object allows you to link an item to a dependency so that the item will be automatically removed from the cache.
- The .NET Framework provides many ways to track user state, including cookies, sessions, and view state. Cookies tend to be reliable, but the browser has to send any information you store in a cookie with every request. Sessions are less reliable and can be used only for temporary information. ASP.NET uses view state to track server control properties between page requests.
- By default, ASP.NET stores session state information in memory. If you deploy your application to multiple web servers, you should store session state information in a state server or a SQL Server database. By default, ASP.NET uses cookies to identify clients with a specific session. If clients do not support cookies, you can choose cookieless sessions instead. However, cookieless sessions add the session ID to the page URL, which can cause a variety of issues.
- View state can significantly increase page size. You can reduce view state page size requirements by using a custom page state persister. ASP.NET provides a built-in page state persister that uses session state; however, it only works reliably if users access only one page at a time.

Objective Review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are designing a web application that will be deployed across ten servers in a web farm. The servers will be load-balanced using NLBS. Sessions must be maintained if a user is directed to a different server mid-session. Which session type should you use? (Choose all that apply. Each answer forms a complete solution.)
   - **A.** InProc session state
   - **B.** State server session state
   - **C.** SQL Server session state
   - **D.** Cookieless session state
2. You recently deployed an ASP.NET web application. Although the application performs well on the intranet, mobile users complain that pages take too long to load. You investigate the problem and determine that the size of the view state is very large, increasing page size. How can you solve the problem? (Choose all that apply. Each answer forms part of the complete solution.)
   - **A.** Create a custom *PageAdapter* class that uses *SessionStatePersister*.
   - **B.** Create a custom *PageAdapter* class that uses control state.
   - **C.** Disable view state.
   - **D.** Use dynamic data controls.
3. Recently, systems administrators scaled a web application you designed from one web server to three web servers. Requests are distributed between the servers using round-robin DNS. Since the upgrade, the administrators have noticed that sessions are reset when a user request is sent to a different server. How can you solve the problem? (Choose all that apply. Each answer forms part of the complete solution.)
   - **A.** Use the *StateServer* session state mode.
   - **B.** Use cookieless session state.
   - **C.** Create a custom *PageStatePersister*.
   - **D.** Configure all web servers with the same machine key.

THOUGHT EXPERIMENT
Scaling State Management

In the following thought experiment, you apply what you've learned about the "Design State Management" objective to predict how a theoretical website architecture will perform. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a developer for Northwind Traders. You are working with management to assess how an Internet application's current state management techniques will scale as they grow from one web server to four.

The application was created by using ASP.NET 4.0. The following aspects of the application rely on state management:

- **Shopping carts** User shopping carts are stored in InProc session state. Administrators increased the session timeout to one hour.
- **User information** User names and addresses are stored in a SQL Server 2008 R2 database. Users are allowed to shop without logging on, but they must log on before checking out.
- **Inventory** Inventory is stored in the database. When a user browses a list of inventory, the web application stores the collection of inventory in the *Cache* object.
- **Support chat status** When support staff are available, the web application provides a link users can click to chat in real time with a support technician. The web application allows support staff to specify whether they are available by adjusting the *Boolean Application["OnlineSupport"]* object.

Northwind Traders has systems administrators on staff. However, the company outsources its web development. As a result, the company would like to minimize the amount of development time needed. Answer the following questions about the change in physical implementation:

1. Which features will require developer time to work properly in the web farm, and why? About how many hours of development effort will be required?
2. What changes will the administrators have to make so that the other features work in a web farm environment?

Chapter Summary

- Client-side scripting provides users with a rich interface, but it lacks the robustness and security of server-side code. SoC simplifies development, testing, and the updating of large-scale web applications by dividing the functional layers of an application. To perform a long-running request while appearing responsive to the user, divide a request into multiple asynchronous steps.
- The two most commonly used system architectures are MVC and three-tier. MFC allows different applications to communicate with a variety of bindings that provide different levels of features, performance, and compatibility. Unless you need multiple servers for redundancy, scalability, or isolation, use a single server. Whenever possible, use coding best practices that separate cross-cutting concerns into separate classes and store settings administrators might want to change separately from your code.
- JavaScript is the only widely accepted client-side scripting language, and it can be extended with the jQuery and Microsoft AJAX libraries. When you need more capabilities than JavaScript can provide, use a plug-in such as Flash or Silverlight. However, you will need to write extra code to provide down-level functionality for browsers that do not support the plug-in.
- HTML controls provide high performance with minimal overhead, but server controls provide much easier access to their properties and values. Custom server controls require extra development effort but provide the most flexibility. Use HtmlHelper extensions to add custom HTML elements to MVC views. To improve page responsiveness, update individual elements within a page using Microsoft AJAX page methods.
- The *Application* and *Cache* objects provide dictionaries that can be accessed by any page and session. You can use cookies and sessions to track information about individual users. ASP.NET stores view state in hidden fields within a page, allowing it to recall server control properties when a page is posted back.

Answers

Objective 1.1: Review

1. **Correct Answer:** C
   - **A. Incorrect:** This approach would display the link only to administrators. However, because the processing is performed on the client, users could view the page source, see that the link existed (even though the browser would not render it), and identify the page it linked to.
   - **B. Incorrect:** Setting the *Visible* property to *False* on the ASPX page would prevent ASP.NET from rendering the control at all. Therefore, JavaScript would not be able to display the control.
   - **C. Correct:** This approach displays the link only to users who are members of the Administrators group, even if the user views the page source. Because it performs the processing on the server, security is maintained.
   - **D. Incorrect:** This approach would never display the link to a user, even if the user was a member of the Administrators group. Additionally, the *HyperLink* control does not have a *Click* event. To initiate a postback when a user clicks a link, you would need to use the *LinkButton* control.
2. **Correct Answer:** A
   - **A. Correct:** This approach blends server-side and client-side programming to provide a responsive user interface. The web server communicates directly with the server hosting the virtual machines because the client would not have authorization.
   - **B. Incorrect:** Although this approach would work, the server would not return a response to the user until after all virtual machines had been provisioned. The lack of responsiveness would be unacceptable to most users.
   - **C. Incorrect:** This approach would work only if the server hosting the virtual machines allowed provisioning requests from the client. However, the server hosting the virtual machines allows provisioning requests only from the web server.
   - **D. Incorrect:** This approach would work if you use JavaScript to initiate web service requests rather than page requests. However, you should not use JavaScript to request webpages unless you intend to display them.

3. **Correct Answer:** B

   A. **Incorrect:** ASP.NET 4.0 web application projects do not allow individual classes to be easily testable. Additionally, ASP.NET 4.0 web applications provide no inherent SoC.

   B. **Correct:** The MVC architecture provides SoC, which also allows individual classes to be testable.

   C. **Incorrect:** Silverlight is not designed to create websites.

   D. **Incorrect:** Like ASP.NET 4.0 web applications, ASP.NET 4.0 websites do not allow individual classes to be easily testable and provide no inherent SoC.

Objective 1.1: Thought Experiment

1. Bandwidth is not a concern because the site is located on a high-speed intranet, and no client-side processing is performed. Therefore, server processing must be limiting the performance of the site. To improve the performance, increase the server's processing capabilities by upgrading the server hardware or adding more servers.
2. With page load times of more than five seconds, the site is already performing unacceptably slowly during peak hours. Adding traffic would cause it to be even slower.
3. You could use Performance Monitor to record the processing time and bandwidth under the current load, and then multiply that by the expected load to estimate the total requirements. To be even more accurate, you could take the site offline and use load-testing software to generate artificial load on the server, and record the performance information by using Performance Monitor.
4. You could generate the charts on the client rather than on the server.
5. You could either generate the charts on the client or use AJAX to replace the chart without reloading the webpage.
6. Generating charts on the client would reduce the processing requirements of the server. You could use JavaScript, Silverlight, or Flash to render the charts on the client. You could provide the raw data to the client by embedding it as an array in the page itself, or you could create a separate web service and have the client chart component retrieve the data from the web service.
7. Because the website provides per-customer information, caching the webpages and charts would have no impact on bandwidth and would actually increase page load times. However, you could reduce bandwidth usage and improve page load times by caching static embedded objects, such as the website logo, JavaScript, Flash, and Silverlight objects.

Objective 1.2: Review

1. **Correct Answer:** C
   - **A. Incorrect:** Refer to answer C.
   - **B. Incorrect:** Refer to answer C.
   - **C. Correct:** To allow the website to remain online if a web server is restarted, you need to configure two web servers. To isolate the database from a denial-of-service attack, you must place it on a separate, third server.
   - **D. Incorrect:** Refer to answer C.
2. **Correct Answer:** A
   - **A. Correct:** The website needs to serve six requests per minute, which is an average of one request every ten seconds. Although the amount of processing time per request varies based on the application, very few applications have requests that take more than ten seconds to process, even on very low-end hardware. The database storage requirement can be met by a single hard disk in a single server. There is no compelling reason to physically separate the web and database servers, so a single server for both roles is the best use of the hardware budget.
   - **B. Incorrect:** Refer to answer A.
   - **C. Correct:** Refer to answer A.
   - **D. Incorrect:** Refer to answer A.
3. **Correct Answer:** A
   - **A. Correct:** Although you could use *WSDualHttpBinding*, *NetTcpBinding* is more efficient.
   - **B. Incorrect:** *NetNamedPipesBinding* is intended for communications between processes on the same computer.
   - **C. Incorrect:** *WsHttpBinding* does not support two-way (duplex) communications.
   - **D. Incorrect:** *WSDualHttpBinding* meets all the requirements; however, *NetTcp Binding* is more efficient.

Objective 1.2: Thought Experiment

1. The company will need to change from *NetNamedPipesBinding* (which only works between processes on a single server) to *NetTcpBinding*. If the developer stored the binding configuration in the .config files, this might only require updating the configuration files. If the developer hard-coded the bindings, that code will need to be edited, but it should be a simple change.
2. The company will need to add a secondary binding type to the logic tier, such as *Net TcpBinding*.

3. The company will need to change only the connection string for the data assembly.
4. The company will need to change the logic tier's binding type to an open-standards based binding type, such as *WsHttpBinding*. It will have a minor, but perhaps not noticeable, negative impact on performance.
5. It's impossible to say without testing the application to determine how many resources each request requires, how many requests each visitor makes, and how quickly visitors send requests at peak hours. However, most low-end servers can handle tens of thousands of visitors per day.

Objective 1.3: Review

1. **Correct Answer:** C
   - A. **Incorrect:** Although a Flash chat client could provide a very rich and responsive user interface, it would be incompatible with some browsers.
   - B. **Incorrect:** As with answer A, Silverlight is incompatible with some browsers.
   - C. **Correct:** Responsiveness is key here, and this approach relies on client-side JavaScript and a separate server process to manage the communications. Because the chat client is running entirely in JavaScript, the server could return a page immediately without waiting for customer service.
   - D. **Incorrect:** Although this approach would work, the server would not return a webpage until after customer service typed a response. Most users are not willing to wait several minutes while a page loads.
2. **Correct Answers:** B and D
   - A. **Incorrect:** Silverlight requires client-side installation.
   - B. **Correct:** jQuery, a JavaScript plug-in, meets all these requirements.
   - C. **Incorrect:** VBScript does not work with all common browsers. It natively works only with Internet Explorer, limiting its usefulness in Internet applications.
   - D. **Correct:** Microsoft AJAX, a JavaScript plug-in, meets all these requirements.
3. **Correct Answer:** A
   - A. **Correct:** Of these choices, only Silverlight provides GPU acceleration. Flash also provides GPU acceleration.
   - B. **Incorrect:** jQuery works with all common browsers and can be used for simple animations. However, at the time of this writing, it does not support GPU acceleration.
   - C. **Incorrect:** VBScript lacks GPU acceleration and natively works only with Internet Explorer.
   - D. **Incorrect:** Microsoft AJAX works with all common browsers and can be used for simple animations. However, at the time of this writing, it does not support GPU acceleration.

Objective 1.3: Thought Experiment

1. Their bandwidth usage will increase for two reasons. First, every client who visits the site will need to download jQuery from the server. Second, clients who visit the home page will need to download the relatively large 500 KB Silverlight object.
2. The easiest way to reduce bandwidth usage is to reference the jQuery library from a CDN, such as that provided by Microsoft. This will also improve page load times for many users. The Silverlight object is very large relative to the size of a typical page; if they reduce the size of that object or remove it entirely, bandwidth usage will be much lower.
3. JavaScript and jQuery are widely accepted, so they should not pose a problem. However, any browser that does not have Silverlight installed (including mobile devices that cannot currently support it) will be unable to view the virtual tool. This is an easy problem to work around; simply determine whether the browser supports Silverlight, and provide alternate content to browsers without the necessary plug-in.

Objective 1.4: Review

1. **Correct Answers:** A and C
   - A. **Correct:** Set the *CssClass* property (and, optionally, the *LoginButtonStyle.CssClass*, *TextBoxStyle.CssClass*, *LabelStyle.CssClass*, *TitleTextStyle.CssClass*, and *ValidatorTextStyle.CssClass* properties) to specify the class assigned to the rendered control in HTML. A web designer can use this class name to associate a set of styles with the control in a CSS style sheet.
   - B. **Incorrect:** When *RenderOuterTable* is set to *true*, ASP.NET uses an HTML table to format the control, rather than allowing the control to be formatted by using CSS style sheets.
   - C. **Correct:** Set *RenderOuterTable* to *false* to allow a control to be styled by using CSS style sheets.
   - D. **Incorrect:** The *LoginButtonType* property controls whether the button is styled as a button, an image, or a link. Setting this value does not help you meet the requirements.
2. **Correct Answer:** D
   - A. **Incorrect:** Partial classes allow you to add functionality to an auto-generated class. They do not provide any client-server communication capabilities.
   - B. **Incorrect:** An *UpdateProgress* control displays the progress of updates to an *UpdatePanel* control. It does not allow you to retrieve string values from a server method.

C. **Incorrect:** You could display an updated string value by using an *UpdatePanel* control. However, the messages would be much larger than if you used a Microsoft AJAX page method.

D. **Correct:** A Microsoft AJAX page method allows you to call server methods without transmitting view state. This minimizes the size of the messages sent between the client and the server.

3. **Correct Answer:** B

A. **Incorrect:** Although you could use jQuery to insert custom HTML into a page after it is rendered, the requirements specify that the control should be rendered on the server.

B. **Correct:** HtmlHelper extensions allow you to create custom Html methods that render HTML code as strings. They are rendered on the server but do not require view state, so they are relatively efficient.

C. **Incorrect:** Custom server controls provide a great deal of flexibility; however, you can meet the requirements by using an HtmlHelper extension, and an HtmlHelper extension requires less development time.

D. **Incorrect:** Custom Web Parts require extra development and consume more client and server resources than an HtmlHelper extension.

Objective 1.4: Thought Experiment

1. Although there are many ways to display the information, a user control is a good choice if it provides enough flexibility to meet the company's requirements. User controls are easy to develop, and it would be simple to add multiple instances of a user control to the *UpdatePanel* container, based on which projects a manager wants to monitor.

2. An *UpdatePanel* and *Timer* will work well to automatically refresh a portion of the page on a regular interval. However, it probably won't be the most efficient way. A more bandwidth-efficient technique would be to create client-side JavaScript that made web services requests (such as page methods) to a server method to retrieve the raw data about a project, and then render it on the client. This technique reduces bandwidth usage and server load, but increases development time. Additionally, although this technique requires only a few lines of server-side code, it would require dozens of lines of client-side JavaScript, which tends to be more difficult to troubleshoot and maintain. Therefore, the trade-off is not entirely clear, and management would need to weigh the benefits of reduced bandwidth against the additional development costs.

Objective 1.5: Review

1. **Correct Answers:** B and C
   - **A. Incorrect:** InProc session state stores session information in the web server's memory. If a user is sent to a different web server, the web server will create a new session for the user.
   - **B. Correct:** When you use state server-based session state, the web servers store and retrieve session state on a central server. All web servers have access to the same session state information, allowing users to move between servers mid-session.
   - **C. Correct:** When you use SQL Server–based session state, the web servers store and retrieve session state on a database server. All web servers have access to the same session state information, allowing users to move between servers mid-session.
   - **D. Incorrect:** Cookieless session state allows browsers that do not support cookies to participate in a session. It does not provide for centralized session state management, however.
2. **Correct Answers:** A and C
   - **A. Correct:** *SessionStatePagePersister* stores view state data within the session on the server instead of using hidden fields that must be sent back and forth between the client and server. *SessionStatePagePersister* can be problematic in some applications, but if users open only one page at a time, it can work properly.
   - **B. Incorrect:** Control state stores data in hidden fields, just like view state. Therefore, using control state would not change the size of the pages.
   - **C. Correct:** Disabling view state would completely remove the hidden fields that increase page size. However, it might prevent the application from working properly. Additionally, disabling view state does not disable control state.
   - **D. Incorrect:** Dynamic data controls use view state by default.
3. **Correct Answers:** A and D
   - **A. Correct:** By default, ASP.NET uses the InProc session state mode, which stores session state information in memory. Although this is very efficient, it does not allow multiple web servers to share session state. You should use StateServer or SqlServer session states any time you use multiple web servers and store information in session state.
   - **B. Incorrect:** Cookieless session state changes how clients identify the session that they are currently using. However, it does not change how the server stores that session data.

C. **Incorrect:** A custom *PageStatePersister* could change the default view state storage mechanism. However, the default view state mechanism of storing data in hidden fields would work properly with multiple web servers and does not need to be changed.

D. **Correct:** Whether you use StateServer or SqlServer session state, you need to synchronize the machine key on all web servers.

Objective 1.5: Thought Experiment

1. The only feature that will require developer time is the support chat status. Because each web server has an isolated instance of the Application object, changing the status would require either updating the *Application["OnlineSupport"]* object on all four servers or writing code to store the support chat status in the database server. Either approach should take only an hour or two of development time (not including testing).
2. Administrators will not need to make any changes to the user information or inventory features; they should work fine in a web farm environment. Naturally, thorough testing will be required to be confident, however. Each web server will maintain a separate cache, but developing a distributed cache might not be worthwhile. They will need to change the session state configuration to use either a state server or the SQL Server, and then update the machine key on all servers.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

About the Author

TONY NORTHRUP, MCPD, MCITP, MCSE, and CISSP, is a consultant and author living in Waterford, Connecticut. Tony started C++ and assembly programming long before Windows 1.0 was released, but has focused on Windows development and administration for the last eighteen years. He has about 30 books and several video training courses covering Windows development, networking, and security. Among other titles, Tony is coauthor of the Microsoft Press MCTS Training Kit for exam 70-515 (Web Applications Development with Microsoft .NET Framework 4) and the author of the Microsoft Press MCTS Training Kit for exam 70-536 (Microsoft .NET Framework Application Development Foundation). You can learn more about Tony by friending him on Facebook at *http://facebook.com/tony.northrup*, visiting his personal website at *www.northrup.org*, reading his technical blog at *http://vistaclues.com*, and viewing his photography portfolio at *http://northrupphotography.com*.