**Controlling Access and Monitoring**

**•** More IIS Security

**•** Using UserID/Password Security

**•** Building a Secured Site

**•** Monitoring Your Site

**•** Disabling an Application for Maintenance

One of the problems with Web applications is that they’re public. By default, anyone who knows the URL can access the site. Another problem is that, by default, you don’t know who’s accessing the site. For many applications that’s not a problem; you want users to access the information. But for others, such as banking and financial transactions, human resources information, insurance, and corporate information systems, you need to know who’s using the site and, in many cases, you want to limit access to a specified group of individuals. You probably want to set up applications that are otherwise open so there are certain areas—for example, usage reports, database alteration operations, content change operations—that application administrators can access, but other users cannot.

**More IIS Security**

**•** Anonymous access, in which case IIS impersonates the IUSR\_MACHINENAME account, which belongs to the Guests group, and thus has, by default, limited access to machine resources. You would normally use this type of access for unsecured applications, and for applications where you manage security yourself using UserID/Password security. If you allow anonymous access, all other types of authentication take effect only if the server denies access to a resource, such as a secured file. To force security across the entire application, you must turn off Anonymous Access.

**•** Basic Authentication works with all browsers. When a user requests a page in a site with Basic Authentication turned on, the server denies the request, which forces the browser to display a UserID/Password dialog. The browser sends the entered UserID and password across the network in plain text, which means the information is insecure. This is the only integrated security option that works with Netscape and other browsers in IIS4. The server checks to see if the information represents a valid account and whether that account has access to the resource. If so, it allows access; otherwise, it returns another Access Denied message.

**•** NT Challenge/Response works only with IE browsers. When a user requests a page from an application with NT Challenge/Response turned on, the server denies the request, just as with Basic Authentication, but the server also returns a challenge. IE recognizes that the server is an NT server and answers the challenge with a hashed value. The server recognizes that the hashed value represents a specific account and checks to see if that account has access to the resource. If so, it allows access, otherwise it returns another Access Denied message. Because the server only has access to the hash value, not to the actual UserID and password, NT Challenge/Response doesn’t allow access to resources on other servers in the network, because the Web server can’t provide the UserID and password when challenged by other servers that have secured resources. NT Challenge/Response also does not work across firewalls.

**Using UserID/Password Security**

The idea behind UserID/Password security is to limit access to all or specific parts of an application. For example, you may have a human resources application where people can change their personal information related to paycheck deposits, insurance or 401K benefits, or apply online for internal company positions. For all these types of applications, you want to ensure that only the person to whom the information belongs can view the information. To do that, you must identify the user.

Just as with other network applications, you need a token of information to identify the user. Typically, you manage access to network resources through a combination of a UserID and password. To acquire the UserID and password, you can build your own form, or you can use the UserID/Password dialog built into the browser. In either case, when you acquire the information, you’ll need to look up the UserID in a database table and verify the password. You’ll need to decide in advance what kind of information to accept. For example, you may allow people to select their own UserID and password, or you might provide them. You may decide to set a minimum length, or to require mixed numbers, letters, and symbols, and you can force a case-sensitive match or disregard case.

**Monitoring Your Site**

In addition to counting pages or total uses of the application, you might want to know other meta-information about your site, such as who’s currently accessing the site, how many requests were denied, where specific users go, or how users navigate to specific pages. Application meta-information provides you with ammunition to justify the cost of the application. An application that costs thousands of dollars to develop might pay for itself in just a few weeks if you can show that it saves money by providing timely information or by reducing the time necessary to accomplish a task.

**Tracking Application Variables**

There are several ways to track site usage. Which one you select depends entirely on the accessibility of your server and your personal preferences—all of them work well. If you’re in an intranet situation and you have Allow Anonymous Access turned off, the simplest method for tracking site usage is to analyze the server log files. With any other type of authentication, IIS logs the UserID of the account that requested the page.

**Planning Applications**

**•** Define the Audience

**•** Determine Application Requirements

**•** Create a Vision

**•** Plan the Interface

**•** Plan Database Requirements

**•** Plan Object Responsibilities

**•** Create Data-Access Components

**•** Create Business Components

**•** Test the Application

**•** Create a Deployment Plan

**•** Create an Application Support Plan

**Define the Audience**

Before you begin any application you should consider the audience. You may or may not have a well-defined target audience. By well-defined, I mean you may or may not know who the users are, what their average reading level is, how much experience they have with computers or with the subject of the application, and what equipment they have to run your application.

For example, if you’re building a browser-based, front-end application to a mainframe application, you probably know, or can find out, exactly who the end-users are. You probably know which browser and which version they’ll be using. You can find out the speed of the local machines. You can visit and talk with them to find out which features they would like, which features they absolutely have to have, and you can use them during development to try out your application, allowing you to make changes before deployment. However, if you’re building an Internet application, you may have no control over the end-users. You’ll need to plan for multiple browsers; unknown versions; slow computers; people who turn off graphics, change fonts, and have varying screen resolutions and color depth.

No application, no matter how well written, can be successful unless it meets the needs of the people who use it. For that to happen, you must determine what those needs are before you write the application.

**Determine Application Requirements**

ASP applications require resources: network resources, server resources, browser resources, and memory, database, security, and support and maintenance resources. All of these topics fall into the category of application requirements. For small test applications where you control the server, you can mostly ignore these requirements, but for larger applications, and those where you do not control the server, you’ll need to do some advance planning.

How big is your application? How many programmer hours will it require? What about other support like subject-matter experts, system analysts, graphic artists, quality control personnel, technical writers, network and server support personnel, and database administrators? Human resources are the single greatest expense in building an application—and in most cases, people underestimate the costs involved in creating them. Coordinating human resources to deliver an application can be a difficult task, especially when rollout deadlines are imposed externally.

Human resources are only one aspect of application requirements

**Create a Vision**

You’ll find that as you determine the audience and application requirements, you build a vision of the application that you can communicate to the various people involved. Use that vision to let others understand why you need the application and what it will do. The goal is to create a vision that others can share; if they can share your vision, they can help you make the application a reality. Without a shared vision, you’ll be ard-pressed to co-opt others into working with you effectively.

|  |
| --- |
| **Plan the Interface**  The interface is the public persona of your application. Some applications have no interface requirements—in fact parts of your ASP applications, as you begin writing data-access and business components, won’t need an interface. However, those portions that do require an interface are the ones that people see. You need to groom and plan them artistically and aesthetically because if people don’t like the interface, they’re unlikely to use the application.  By the terms artistically and aesthetically, I don’t mean you should add graphics or try to disguise the purpose of the application in favor of making it beautiful, I mean you should pay attention to the way you lay out controls, images, and text on the page. Take the time to align and size controls properly. Don’t select a gray background color for text input fields—people think they’re disabled or locked and won’t try to type in them. Don’t create a text field 200 characters wide if people are only going to enter 10 characters into the field. Similarly, leave enough room between controls. I’ve seen numerous applications—especially data-entry applications—where the designer has packed so many controls onto the screen that it’s impossible to pick out the important features.  **Plan Database Requirements**  ASP and Access have made it nearly painless to get a database application up and running, but for any substantive application, that’s neither complete nor sufficient. When you build a database, you probably plan to gather information. Unless your application is so completely isolated that no one else wants to interact with the information you gather, it’s highly likely that others will eventually need to interact with the database as well.  The database is the backbone of your application; therefore, it should be bulletproof. That means it should not accept information that doesn’t meet requirements. It’s easy to write code that tests for information validity, but you can’t be sure everyone will write bulletproof code. While you would never allow an invalid value in a field, you can be sure that eventually one will slip through someone else’s code.  Databases on the Web must service all the application’s users, and because of the stateless nature of the Web, they are called upon to provide more information much more often than would normally be required in a standard client-server situation. Because database operations are inherently machine-and network-intensive, it’s your job to minimize the volume of data that must traverse the network for any given request. That means your task is to learn to take advantage of the power of SQL to select only the required data. But that’s only the beginning. |

**Plan Object Responsibilities**

If databases are the backbone of a modern Web application, objects are the legs. Objects in a Web application are often transient—created, used, and destroyed as quickly as possible. Consider the ASP intrinsic objects. Only the Application object is permanent; IIS creates and destroys all the others for each request. That’s a lot of objects. Does the ASP engine really create all the intrinsic objects for each request or does it create them only when you write a reference in your code? Quite truthfully, I don’t know, except for the Request.ServerVariables collection. The Microsoft documentation states that ASP creates the Request.ServerVariables collection only if you reference it, because the overhead involved in creating the collection is relatively high. In any case, the ASP object model has stood the test of time—it has remained relatively static ever since version 1. IIS4 introduced transactional capabilities and added the ObjectContext object. IIS5 introduced the Server.Transfer and Server.Execute methods, as well as the Session and Application Contents collections; but most methods and properties remain unchanged.

**Create Data-Access Components**

The most scalable Web applications use data-access objects to request data from and update database tables. That’s not the most efficient way to develop your application—it’s much easier and faster to write dynamic SQL queries directly into your ASP code than to plan and develop COM database components. Nevertheless, dynamic SQL queries can be used only by your application, whereas COM components may be used by many different applications. Data-access components become much more important when your application isn’t the only application using the database.

The purpose of a data-access component is to mediate data transfer to and from the database. These components often run inside of Microsoft Transaction Server (MTS), which instantiates objects on demand, rather than on request. That means any given object may be shared between many different clients.

**Create Business Components**

Business components contain the business rules that act on the data provided by data-access components. You often want to isolate these rules in a separate layer because they change more often than the front-end interface requirements. Consider an application that arranges route patterns for salespeople. The route data—the customer list and locations, resides in a database. Although the data changes constantly, the format of the data does not. You might build a data-access component to retrieve and update the data. On the other hand, the rules for calculating the route change constantly, depending on pricing, availability, the potential size of orders from different customers, and many other business factors.

**Test the Application**

Development is often a tradeoff between efficiency and stability. While you may win praise for delivering an application on time, such praise will rapidly turn to censure if the application fails to work as anticipated. Therefore, don’t forget to plan for testing. In my experience, there are, at minimum, three testing phases for an application. Each testing phase should answer one or more questions. The phases are:

Programmatic testing does the application work according to specifications? This phase begins when code development begins and continues throughout the development of the application. As the program moves from development into maintenance, you must repeat this phase for every change to the application.

**Create a Deployment Plan**

Designing, building, testing, and debugging applications are the fun parts of program development. Deployment is rarely as satisfying because deployed machines are rarely completely under your control. You’ll have the best chance of success, and deployment will be less painful if you follow these guidelines.

**Prepare for Deployment**

Deploying an ASP application isn’t much different from deploying a standard Windows application, but there are three critical differences:

1.  Anonymous Web applications run under different permissions than standard applications. Web applications using NT Challenge/Response run under the user’s account on the server, but cannot authenticate across the network.

2.  Web applications run on multiple threads. Resource contention that you may not see during development can become an issue after deployment.

3.  The production server will not be your development server, won’t have the same directory permissions that your server does, may not be the same service pack version, and will have different security settings. If you’re lucky, you’ll have administrator access to the production server. In both cases, you can help either yourself or the Server Administrators by following the guidelines in this chapter.

**Create an Application Support Plan**

After deployment you should be prepared to receive problem calls from end-users. Sometimes such calls are questions about how to activate or use the application, but sometimes they’re problem reports. Unless you plan to take the calls yourself, you need to provide program documentation to the Help desk or to the people who work with the end-users to solve the problem. The work you did to document error messages and program requirements during the deployment phase should form the basis of your application support plan.

|  |
| --- |
| **Using ActiveX Controls**  Sometimes, even with DHTML, the capabilities available through HTML and a browser just aren’t enough. For example, suppose you want to use a slider control to let a user control audio volume. You could create your own slider using lines or images and a great deal of code, but in doing so you would have recreated functionality that’s already available on every Windows computer in the form of ActiveX controls. Recognizing this problem, Microsoft did two things to IE that made it the premier browser for development. First, IE is an ActiveX host. That means you can site (place) ActiveX controls within the browser and control them via script. Second, Microsoft made IE itself an ActiveX control. That means you can site IE within a window and instantly gain access to a custom browser for your own programs. In this section, you’ll see how to use ActiveX controls inside IE.  You’ve already seen how to instantiate ActiveX components on the server. The client-side analogues of ActiveX components are ActiveX objects or ActiveX controls. All three—ActiveX components, objects, and controls—are COM objects. The only difference is that ActiveX controls are usually visible so people can interact with them directly, whereas components and objects are usually invisible. To instantiate ActiveX controls within the browser, you use an <object> syntax that’s similar to the syntax you’ve already used to create Application- and Session-scope components in the Global.asa file. For example:  **<object**  **classid=“clsid:25bdf09d-ec8b-11cf-bd97-00aa00575603”**  **codebase=“/Controls/MyContrl.cab#version=1,0,0,0”**  **id=myControl left: 40px; top: 10px; width: 385px;**  **height: 42px; >**  **<param name=“property1”>**  **<param name=“property1”>**  **<param name=“property1”>**  **</object>** |